ALESSANDRO RODRIGUES ZAMBONI

UMA PROPOSTA DE SISTEMA DE SOFTWARE PARA AUXÍLIO NA GERAÇÃO DE TRANSFORMAÇÕES DO FORMALISMO BAV PARA INTEGRAÇÃO DE BANCOS DE DADOS

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. André Luiz Pires Guedes

CURITIBA

2004

ALESSANDRO RODRIGUES ZAMBONI

UMA PROPOSTA DE SISTEMA DE SOFTWARE PARA AUXÍLIO NA GERAÇÃO DE TRANSFORMAÇÕES DO FORMALISMO BAV PARA INTEGRAÇÃO DE BANCOS DE DADOS

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. André Luiz Pires Guedes

CURITIBA

2004

ALESSANDRO RODRIGUES ZAMBONI

UMA PROPOSTA DE SISTEMA DE SOFTWARE PARA AUXÍLIO NA GERAÇÃO DE TRANSFORMAÇÕES DO FORMALISMO BAV PARA INTEGRAÇÃO DE BANCOS DE DADOS

Dissertação aprovada como requisito parcial à obtenção do grau de Mestre no Programa de Pós-Graduação em Informática da Universidade Federal do Paraná, pela Comissão formada pelos professores:

Orientador: Prof. Dr. André Luiz Pires Guedes
Departamento de Informática, UFPR

Prof. Dr. Milton Ramos Ramirez
Instituto de Matemática, UFRJ

Prof. Dr. Marcos Sfair Sunye
Departamento de Informática, UFPR

Curitiba, 7 de julho de 2004

Ministerio da Educação
Universidade Federal do Parana
Mestrado em Informatica

PARECER

Nos, abaixo assinados, membros da Banca Examinadora da defesa de Dissertação de Mestrado em Informatica, do aluno *Alessandro Rodrigues Zamboni*, avaliamos o trabalho intitulado, "*UMA PROPOSTA DE SISTEMA DE SOFTWARE PARA AUXILIO NA GERAÇÃO DE TRANSFORMAÇÕES DO FORMALISMO BAV PARA INTEGRAÇÃO DE BANCOS DE DADOS*", cuja defesa foi realizada no dia 07 de julho de 2004, as dez horas, no Auditorio do Departamento de Informatica do Setor de Ciências Exatas da Universidade Federal do Parana Apos a avaliação, decidimos pela aprovação do candidato

Curitiba, 07 de julho de 2004

Prof Dr Andre Luiz Pires Guedes
DINF/UFPR – Orientador

Prof Dr Milton Ramos Ramirez
UFRJ – Membro Externo

Prof Dr Marcos Sfair Sunyê
DINF/UFPR – Membro Interno

AGRADECIMENTOS

Agradeço à Coordenação de Graduação e Pós-Graduação de Informática da Universidade Federal do Paraná pela educação e confiança depositadas em mim ao longo desses anos. Agradeço a meu orientador, Prof. Dr. André Luiz Pires Guedes, pelo apoio e pelos sábios conselhos nos momentos de desespero. Agradeço aos professores Marcos Sunye e Carmem Satie Hara por me mostrarem a direção a seguir. Por fim agradeço a todos os meus colegas de mestrado e a meus familiares pela pequenas contribuições que fizeram a diferença entre a vitória e o fracasso.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

O objetivo da integração de bancos de dados é fornecer uma interface integrada e uniforme para consultas em diversos bancos de dados que descrevam os mesmos objetos do mundo real. O formalismo "both as view"(BAV) foi desenvolvido de modo a ser aplicado em esquemas de bancos de dados para realizar a sua integração. Seu funcionamento se caracteriza por ver o processo de integração como uma sequência de transformações reversíveis que modificam tanto o esquema como as instância do banco de dados. Este trabalho apresenta uma proposta de sistema de software que automatiza parcialmente o processo de integração, auxiliando o administrador de banco de dados na tarefa de integração, guiando-o pelo processo de gerar transformações e consultando-o em momentos em que seja impossível para o sistema tomar uma decisão. Além disso são apresentados estudos realizados sobre o formalismo BAV e sua aplicação nos casos de conflito entre esquemas de bancos de dados.

ABSTRACT

The objective of the database integration is to supply an integrated uniform interface to consultations in diverse databases that describe same objects of the real world. The formalism "both view"(BAV) was developed in order to be applied in schemas during the process of databases integration. Its functionality characterizes for seeing the integration process as a sequence of reversible transformations that modify the schema and the instances of the data base. This work presents a proposal of system software that automatizes the integration process partially, assisting the user in the integration process, guiding him to generate transformations and consulting it at moments where the system can not take a decision. Moreover are presented studies about the BAV formalism and its applications in the cases of conflict between database schemas.

CAPÍTULO 1

INTRODUÇÃO

Um banco de dados é construído com o objetivo de armazenar informações de um modo estruturado permitindo consultas rápidas e eficazes sobre as mesmas. O esquema de um banco de dados representa o modo como ele guarda as informações. Suas instâncias são as informações propriamente ditas.

Um dos maiores objetivos da integração de dados é fornecer uma interface integrada e uniforme para consultas em diversos bancos de dados que descrevam os mesmos objetos do mundo real. Assim um sistema de integração de dados possibilita ao usuário se concentrar no que ele deseja sem ter de pensar em como conseguir essa informação.

Desse modo a integração de bancos de dados é o processo que toma como entrada um conjunto de bancos de dados e produz como saída uma única descrição dos esquemas dos bancos de dados de entrada (esquema integrado) e um mapeamento das informações associadas que permita o acesso as informações existentes nos bancos de dados de entrada através do esquema integrado.

O estudo do processo de integração de bancos de dados nas suas diversas formas de abordagem e realização foi feito, entre outros, por C. Parent e S. Spaccapietra em [18] e por C. Batini, M. Lenzerini e S. B. Navathe em [2]. Eles dividiram o processo em fases conforme suas necessidades ou preocupações sugerindo as técnicas mais utilizadas para a sua resolução. De modo sucinto, essas fases podem ser apresentadas como:

- considerações iniciais - fase onde os esquemas a serem integrados são transformados de modo a se tornarem sintaticamente e semânticamente mais homogêneos;

- pré-integração - consiste de uma análise dos esquemas a serem integrados com o objetivo de escolher uma política de integração eficiente, definindo, entre outras coisas, quais esquemas serão integrados e em que ordem ou se serão integrados apenas pedaços dos esquemas ao invés do esquema inteiro;

- identificação de correspondências - fase onde as correspondências entre os esquemas são identificadas e descritas;

- identificação de conflitos - fase onde os esquemas são analizados e comparados para determinar as correspondências entre conceitos e detectar possíveis conflitos (conflitos de tipo, estrutura, etc.);

- integração - fase onde os conflitos entre os esquemas são resolvidos e os itens correspondentes são unificados;

- união e reestruturação - nesta fase são gerados esquemas integrados intermediários, para análise e se necessário, uma reestruturação para que se alcance qualidades desejáveis como completude, corretude, minimalidade e entendibilidade.

Os casos de conflito recebem uma atenção especial neste trabalho, uma vez que a identificação e a resolução dos mesmos está entre os maiores desafios encontrados pelo processo de integração de bancos de dados. Os casos de conflito existentes podem ser divididos em conflitos de incompatibilidade de domínio e conflitos entre definições de entidades, sendo listados a seguir:

- conflito de nomes - ocorre quando dois objetos semânticamente equivalentes possuem nomes diferentes ou quando dois objetos sem nenhuma equivalência semântica possuem nomes iguais;

- conflito de representação dos dados - ocorre quando dois objetos semânticamente equivalentes são representados por diferentes tipos de dados;

- conflito de unidades - ocorre quando dois objetos semânticamente equivalentes são representados em diferentes unidades ou medidas;

- conflito de precisão dos dados - ocorre quando dois atributos que são semanticamente parecidos são representados com diferentes unidades e medidas de modo a não ocorrer uma correspondência de um para um entre os respectivos domínios;

- conflito de valores padrão - ocorre quando em um atributo passa a ter um valor específico quando não é indicado um valor para ele;
- conflito de equivalência de chaves - ocorre quando duas entidades semanticamente parecidas são definidas com identificadores semanticamente diferentes;
- conflito de união - ocorre quando dois objetos semânticamente equivalentes são representados com um número diferente de atributos ou com atributos não relacionados de modo que a união entre eles não é compatível;
- conflito de isomorfismo - ocorre quando um número diferente de atributos é utilizado para representar conceitos similares;
- conflito de falta de atributos - ocorre quando o atributo que falta pode ser recuperado através de mecanismos de inferência;
- conflito entre relaçoes e atributos - ocorre quando o mesmo objeto é modelado como um atributo em um esquema e como uma relação em outro esquema;
- conflito entre atributos e dados - ocorre quando o valor de um atributo em um esquema corresponde ao próprio atributo em outro esquema.

De modo a permitir que o sistema de integração de banco de dados possa responder as consultas, deve haver alguma descrição do relacionamento entre o esquema global e os esquemas locais. O processador de consultas deve estar apto a reformular uma consulta submetida a ele como novas consultas submetidas aos esquemas locais [9]. Dois formalismos de grande importância para a integração de bancos de dados são os formalismos "global as view"(GAV) e "local as view"(LAV). No formalismo GAV, para cada relação pertencente ao esquema global, escreve-se uma consulta sobre os esquemas locais especificando como obter as instâncias destinadas a essa relação. Já no formalismo LAV cada uma das relações das fontes é descrita através de consultas sobre o esquema global.

Através da parceria de pesquisas realizada pela *Universidade de Birkbeck (Birkbeck University of London)* e pela *Universidade Imperial (Imperial College of London)* foi criado o *Projeto de Geração Automática de Ferramentas Mediadoras para Integração de*

Bancos de Dados Heterogêneos (Automatic Generation of Mediator Tools for Heterogeneous Database Integration - AutoMed) [3].

No AutoMed as linguagens de modelagem para bancos de dados são representadas através de um modelo de dados baseado em hipergrafos (HDM) [12]. A metodologia utilizada pelo projeto AutoMed se caracteriza por ver o processo de integração como uma sequência de transformações reversíveis que modificam tanto o esquema como a extensão do banco de dados. Essa sequência de transformações incrementalmente adiciona, apaga ou renomeia um objeto do esquema do banco de dados até que se possa mapear diversos esquemas de bancos de dados uns com os outros.

O formalismo *both as view (BAV)* [13, 14, 11] foi desenvolvido a partir do formalismo HDM e dos formalismos GAV e LAV e desenvolvido de modo a ser aplicado em esquemas que usem o modelo de dados relacional.

Do mesmo modo que o formalismo HDM, o formalimo BAV se caracteriza por transformações aplicadas em sequência onde cada uma adiciona, retira ou renomeia um objeto do formalismo. Além disso são especificadas operações de *extend* e *contract* que respectivamente adicionam ou retiram objetos do formalismo especificando a consulta que determina a origem das instâncias apenas parcialmente ou até mesmo não as especificando.

O Projeto AutoMed possui diversas metodologias para realizar a integração de dados através do formalismo BAV. Todas elas se caracterizam por produzir sequências de transformações aplicadas tanto no esquema quanto nas instâncias de cada uma das fontes de dados. Duas metodologias se destacam por serem as mais utilizadas.

A primeira metodologia [13] se destaca por uma postura mais artesanal, direcionada a casos específicos onde não se pretende adicionar ou retirar fontes de dados do processo de integração após seu término. Esta metodologia dá grande ênfase a completude do processo de integração, incentivando a idéia de que toda as informações das fontes de dados locais devem estar no banco de dados global. Costuma ser utilizada, por exemplo, em casos onde grandes corporações precisam integrar bancos de dados muito extensos.

A segunda metodologia [11] se destaca por uma postura de trabalho como a de uma linha de montagem. Por um lado é mais flexível, uma vez que é razoavelmente fácil adi-

cionar ou retirar fontes de dados ao processo de integração mesmo após seu término. Por outro lado a completude do processo de integração é ignorada uma vez que normalmente nem todas as informações das fontes de dados locais estarão presentes no banco de dados global. Costuma ser utilizada, por exemplo, em casos em que existe a necessidade de integrar muitos bancos de dados de tamanho reduzido como locadoras ou escolas.

1.1 Objetivos

O objetivo desta dissertação é apresentar uma proposta de sistema de software para integração de bancos de dados através do formalismo BAV.

Este sistema de software é aplicado na metodologia de integração que aplica o formalismo nas fontes de dados como se estas estivessem em uma linha de montagem. Nessa metodologia assume-se um esquema de união e cada esquema local terá de se tornar compatível à união com o esquema assumido por meio de transformações BAV.

O objetivo deste sistema de software é auxiliar o usuário na geração das transformações BAV que tornarão a estrutura de um esquema local idêntica ao esquema de união. Para alcançar esse objetivo o sistema orienta o usuário através das várias etapas do processo consultando-o em momentos em que seja impossível para o sistema tomar uma decisão.

De modo sucinto, o que se espera do funcionamento do sistema de software proposto neste trabalho é que sejam fornecidos para ele dois esquemas, o primeiro relativo a um banco de dados local e o segundo relativo ao esquema de união definido para a integração. A medida que realiza diversas consultas ao usuário o software deve fornecer a sequência de transformações necessária para tornar a estrutura do esquema relativo ao banco de dados local idêntica a estrutura do esquema de união. A figura 1.1 ilustra esse processo mais nitidamente. Baseado nessas características de funcionamento o sistema de software foi batizado como: *Sistema de Software para Auxilio na Geração de Transformações BAV para Integração de Bancos de Dados*, abreviado através da sigla *SAI*.

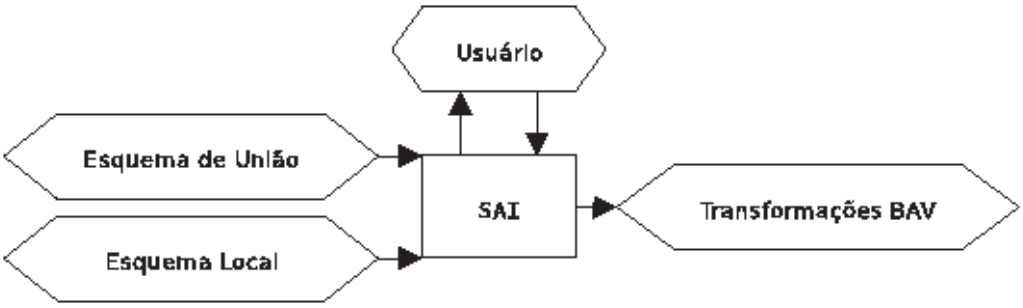

Figura 1.1: Funcionamento do Sistema SAI

1.2 Organização

Esta dissertação está organizada em seis capítulos divididos conforme seus objetivos.

O capítulo dois apresenta uma breve introdução aos bancos de dados, apresentando conceitos básicos utilizados no decorrer da dissertação.

O capítulo três descreve o estudo da integração de dados, dividindo-o em fases, apontando seus problemas e mostrando alguns formalismos utilizados para sua resolução.

O capítulo quatro descreve o projeto AutoMed, o formalismo baseado em hipergrafos, o formalismo BAV, suas metodologias, a linguagem utilizada por esses formalismos para descrever consultas e um estudo feito nesta dissertação sobre como o formalismo BAV pode resolver cada um dos casos de conflito.

O capítulo cinco expõe a proposta do sistema de software SAI, o objetivo desta dissertação.

O capítulo seis conclui esta dissertação, expondo resultados, contribuições e projetos futuros.

Existem ainda dois anexos acompanhando este trabalho. O anexo A define mais formalmente o formalismo baseado em hipergrafos. O anexo B contém especificações informais de cada uma das etapas do sistema de software SAI.

CAPÍTULO 2

BANCOS DE DADOS

É necessário apresentar de modo breve e sucinto a tecnologia de bancos com o objetivo de tornar termos utilizados neste trabalho mais claros. Este capítulo foi baseado em [24, 25, 1, 26, 22].

2.1 Sistema Gerenciador de Bancos de Dados

Para entender o funcionamento de um sistema gerenciador de bancos de dados é necessário ter em mente o ambiente em que ele atua e suas necessidades. Imagine uma grande empresa ou indústria, como uma fábrica de automóveis por exemplo. Tal fábrica possui uma grande quantidade de dados que precisam ser mantidos por um longo período de tempo. Entre outras informações estão fornecedores, estoque, clientes, setores, empregados, produção, contratos, etc. Essas informações se relacionam entre si e como possíveis relacionamentos estão as entregas (quais clientes receberam carros de quais lotes), a lista de compras (quanto comprar de cada fornecedor para manter o estoque constante), distribuição dos empregados (quais empregados trabalham em quais setores), etc.

Dados como esses que são armazenados mais ou menos permanentemente em um computador são chamados de *Bancos de Dados*. O software que permite que uma ou mais pessoas usem e/ou modifiquem esses dados é chamado de *Sistema Gerenciador de Bancos de Dados (Database Management System - DBMS)*. A principal função de um DBMS é permitir que o usuário manipule os dados de uma forma abstrata ao invés de manipular os dados na forma como são armazenados no computador. Nesse sentido o DBMS age como um interpretador para uma linguagem de programação de alto nível, permitindo que o usuário especifique o que precisa ser feito, com pouca ou nenhuma atenção aos algorítmos e representações de dados utilizados pelo sistema.

2.2 Esquemas e Instâncias

Durante a fase de criação de um banco de dados o foco de interesse está no objetivo final do banco de dados. Já na fase de utilização o foco de interesse está nas informações que ele contém. Os dados contidos em um banco de dados mudam frequentemente mas seu objetivo final continua o mesmo por longos períodos de tempo (apesar de não durar para sempre em alguns casos).

Os termos *esquema* ou *intenção* são usados para referenciar os objetivos do banco de dados expressos como uma enumeração dos tipos de entidades com os quais o banco de dados irá lidar e os relacionamentos entre elas. Já o conteúdo de um banco de dados é chamado de *instância* ou *extensão* do banco de dados.

2.3 Modelos de Dados

Um *Modelo de Dados (Data Model)* é um formalismo matemático composto de duas partes: uma notação para descrever os dados e um conjunto de operações para manipular esses dados.

Existe uma grande quantidade de modelos de dados em uso atualmente. Cada um deles com características e objetivos específicos. Entre as qualidades mais relevantes que diferenciam modelos de dados uns dos outros estão:

- propósito;
- orientação a objetos ou valores;
- capacidade de lidar com redundância;
- capacidade de lidar com relacionamentos de alta cardinalidade (*muitos-para-muitos*).

2.3.1 O Modelo Entidade-Relacionamento

O propósito do modelo entidade-relacionamento é permitir a descrição do esquema conceitual sem preocupações com a eficiência do sistema, com a estrutura física do banco de dados ou até mesmo com o DBMS como seria esperado na maioria dos modelos.

Esse modelo de dados carece de um conjunto de operações sobre seus dados. De certo modo pode-se afirmar que ele não é um modelo de dados completo por definição. Apesar disso esse modelo mostra sua utilidade ao ser usado para justificar os tipos de estruturas e modelos de dados que venham a ser utilizados posteriormente na fase de implementação. Este modelo também pode se mostrar muito prático quando existe a necessidade de consultar o usuário sobre assuntos relativos ao projeto.

Uma *entidade* representa algo que exista e seja distinguível, sendo possível diferenciar um elemento de outro, como por exemplo uma pessoa, um carro ou uma loja. Um *conjunto de entidades* consiste de um grupo de entidades similares, como o conjunto de todas as pessoas, todos os carros ou todas as lojas de uma região.

Conjuntos de entidades possuem propriedades, chamadas *atributos*, que associam para cada entidade do conjunto um valor pertencente a um domínio de valores. Normalmente o domínio de valores para um atributo será o conjunto dos inteiros, reais ou strings de caracteres, apesar de que outros tipos de domínios de valores também podem ser utilizados. Uma conjunto de entidades que representam pessoas podem possuir atributos como nome ou peso, por exemplo.

O conjunto de atributos cujos valores identificam de modo único cada entidade são chamados de *atributos chaves*. Cada conjunto de entidades deve possuir um conjunto de chaves de modo a se distinguir uma entidade específica em um conjunto de entidades. Um possível exemplo de chave para o conjunto de entidades pessoa seria um identificador único como o CPF ou o RG.

O termo *ISA*, utilizado na forma A ISA B quando A e B forem conjuntos de entidades, identifica B como uma generalização de A, ou seja, A é um tipo especial de B. Nesse caso, A herda todos os atributos de B, mas possui atributos próprios que B não possui. O conjunto de chaves de A é o mesmo de B de modo que seus valores para entidades correspondente em A e B seja igual. Possíveis exemplos para a generalização veículos seriam carros ou motocicletas.

Diferentes conjuntos de entidades geralmente não têm utilidade em isolamento uns dos outros. Eles são associados uns aos outros através de *relacionamentos*. Por exemplo,

pessoas possuem carros ou pessoas trabalham em lojas.

Relacionamentos costumam ser classificados de acordo com o número de entidades de dois conjuntos de entidades que se relacionam entre si. Existem três classificações possíveis: *um-para-um*, que são os mais simples e mais raros, *um-para-muitos* e *muitos-para-muitos*.

Diagramas entidade-relacionamentos são utilizados para resumir todo o trabalho de uma maneira fácil de apresentar e entender. Estes diagramas utilizam a seguinte metodologia:

- entidades são representadas por retângulos;
- relacionamentos são representados por losangos que são ligados aos retângulos através de retas não direcionadas;
- atributos são representados por círculos que são ligados aos retângulos ou losangos através de retas não direcionadas;
- os nomes dos atributos são sublinhados quando eles forem chaves;

Um exemplo de diagrama entidade-relacionamento é apresentado na figura 2.1.

2.3.2 O Modelo de Dados Relacional

Desde sua apresentação em 1970, o modelo de dados relacional tem crescido em importância até se tornar o modelo de dados mais utilizado no mundo para implementar novos bancos de dados. Provavelmente a maior razão da sua popularidade é a maneira como ele suporta poderosas, apesar de simples, linguagens declarativas através das quais as operações sobre os dados são expressas.

O fato desse modelo de dados ser orientado a valor permite que se possa definir operações sobre relações cujos resultados são também relações. Desse modo estas operações podem ser combinadas sequencialmente em cascata utilizando a notação chamada de *álgebra relacional*.

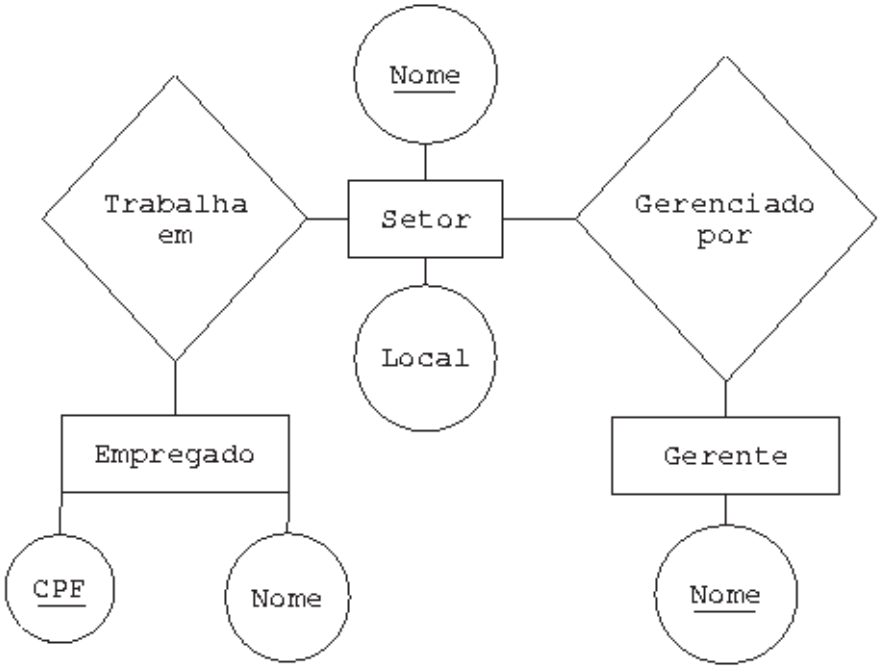

Figura 2.1: Exemplo de Diagrama Entidade-Relacionamento

O conceito matemático no qual reside o modelo de dados relacional é o da teoria de conjuntos. Uma *relação* é qualquer subconjunto do produto cartesiano entre um ou mais domínios. Por exemplo, $\{(0, a), (0, c), (1, b)\}$ é uma relação, um subconjunto de $D_1 x D_2$ para $D_1 = \{0, 1\}$ e $D_2 = \{a, b, c\}$. Cada um dos membros de uma relação é denominado *tupla*, cada um dos membros de uma tupla é denominado *atributo* e o número de membros de uma tupla em uma relação é denominado *aridade*. É importante lembrar que uma vez que cada atributo recebe um nome a ordem dos atributos em uma relação perde a importância. Por outro lado, também é muito comum imaginar uma relação como uma tabela, onde cada linha é uma tupla e cada coluna é um componente da tupla, um atributo.

Tal como conjuntos de entidades no modelo entidade-relacionamento, relações também possuem atributos que funcionam como chaves no modelo relacional. Tais atributos são chamados de *chaves primárias (primary keys - PK)*.

As operações utilizados para manipulação dos dados no modelo relacional são:

- união - representada por $R \cup S$, toma duas relações R e S e devolve o conjunto das tuplas que estão em R, em S ou em ambas. Esta operação só pode ser realizada em relações que possuam a mesma aridade;

- diferença - representada por $R - S$, toma duas relações R e S e devolve o conjunto das tuplas que estão em R mas não estão em S. Esta operação só pode ser realizada em relações que possuam a mesma aridade;

- projeção - dada uma relação R, essa operação remove e/ou reordena seus atributos gerando uma nova relação;

- seleção - esta operação testa para cada tupla na relação R uma expressão lógica L e devolve na forma de uma relação todas a tuplas para as quais L seja verdadeiro.

- produto cartesiano - representada por $R \otimes S$, toma duas relações R e S, com aridade k_1 e k_2 respectivamente, e devolve o conjunto de todas as possíveis tuplas com aridade $k_1 + k_2$ cujos primeiros k_1 elementos pertençam a R seguidos por k_2 elementos pertencentes a S;

A partir das operações citadas pode-se construir várias outras operações. Entre elas estão as operações de quociente, join, join natural, etc.

2.3.3 Outros Modelos de Dados

2.3.3.1 O Modelo de Dados em Rede

Informalmente pode-se definir o modelo de dados em rede como o modelo entidade-relacionamento tendo todos os seus relacionamentos restritos a serem binários ou um para muitos. Esses relacionamentos são chamados de *links* no modelo de dados em rede. No lugar de conjuntos de entidades o modelos de dados em rede oferece *tipos de registros lógicos*. Tipos de registros lógicos são compostos de campos nos quais valores elementares como inteiros ou strings de caracteres são guardados. O conjunto de nomes para esses campos e seus tipos de dados é chamado de *formato do registro lógico*.

Pode-se ver uma analogia entre os termos utilizados no modelo relacional e os termos utilizados no modelo de dados em rede na tabela 2.1.

Contudo há uma importante distinção a ser feita entre os dois modelos. Como o modelo de dados relacional é orientado a valor duas tuplas exatamente iguais com os

Modelo em Rede	Modelo Relacional
Formato do Registro Lógico	Esquema da Relacão
Tipo de Registro Lógico	Nome da Relação
Registro Lógico	Tupla

Tabela 2.1: Analogia entre o Modelo de Dados Relacional e o Modelo de Dados em Rede

mesmos valores para todos os atributos são vistas como a mesma tupla. Já o modelo de dados em rede é orientado a objeto, desse modo todo registro lógico possui um identificador transparente que é o seu endereço e mesmo que dois registros lógicos sejam iguais em todos os seus campos eles não são vistos como sendo o mesmo registro lógico.

A utilização de links no modelo de dados em rede permite a construção de um grafo direcionado chamado de *rede*. Esse grafo não é nada mais que um modelo simplificado do diagrama entidade-relacionamento que tem como objetivo representar os tipos de registros e seus links.

2.3.3.2 O Modelo de Dados Hierárquico

O modelo de dados hierárquico é composto por uma coleção de registros conectados uns aos outros através de links de maneira similar ao modelo de dados em rede. Na verdade pode-se afirmar que o modelo de dados hierárquico é um modelo de dados em rede no qual o grafo resultante lembra um floresta (um conjunto de árvores). Isto é obtido com a construção de um vértice virtual para cada árvores do modelo. Este vértice têm a função de ser a raiz da árvore do banco de dados. Ao contrário do modelo de dados em rede, os links no modelo de dados hierárquico são direcionados de pai para filho restringindo os caminhos possíveis na árvore.

2.3.3.3 O Modelo de Dados Orientado a Objeto

Ao contrário do que se possa pensar, não existe apenas uma mas várias propostas e implementações do que deveria ser um modelo de dados orientado a objeto. Apesar da variação de nomes que eles possam ter (tais como semântico, funcional, etc.) eles reúnem caraterísticas em comum. São elas:

- identidade dos objetos - os elementos trabalhados são registros com um endereço único, seguindo a linha dos modelos de dados hierárquico e em rede;
- objetos complexos - é permitida a construção de novos tipos a partir de tipos nativos;
- hierarquia de tipos - é permitido que tipos tenham subtipos com propriedades especiais.

A idéia básica por traz do modelo de dados orientado a objeto é a de encapsular dados e código relacionados a um objeto em uma simples unidade. Como resultado disso toda a interação entre um objeto e o resto do sistema é feita através de mensagens. Geralmente um objeto é composto por um conjunto de variáveis que contém os dados relativos ao objeto, um conjunto de mensagens para os quais o objeto está programado a responder e um conjunto de métodos que implementam e respondem as mensagens.

Objetos similares, ou seja, que atendem as mesmas mensagens, usam os mesmos métodos e possuem váriaveis com nomes e tipos idênticos, são agrupados de modo a formar uma *classe*. E cada ocorrência da classe é chamada de *instância* da classe.

O conceito de hierarquia de classes permite que uma classe possua um conjunto de subclasses (ou classes filhas) que herdam todas as propriedades da classe pai e possuem características próprias referentes apenas a elas.

Se for feita uma comparação com o modelo de dados entidade-relacionamento, um objeto corresponderia a uma entidade, uma classe corresponderia a um conjunto de entidades, uma subclasse corresponderia a uma generalização do tipo ISA e uma variável corresponderia a um atributo.

2.4 Bancos de Dados Distribuídos

Um banco de dados distribuídos possui seus dados dispersos e/ou replicados em vários locais. Computadores diferentes controlam o acesso a diferentes partes dos dados e as máquinas utilizadas para gerar a inteface com o usuário podem não ser as mesmas que fazem o acesso aos dados. Estes computadores costumam estar ligados por links de comunicação que geralmente possuem uma baixa velocidade em comparação com o resto

do sistema. Consequentemente esses links se tornam o gargalo dos bancos de dados distribuídos e a maior parte dos trabalhos e pesquisas relativos a essa área contribuem com diferentes formas de lidar com esse gargalo.

As relações se dividem em dois tipos: relações lógicas, que não existem de verdade e são construídas virtualmente pela união de fragmentos de relações, e relações reais, também chamadas de relações físicas que constituem os próprios fragmentos de relações.

2.5 Conclusão

Nesse capítulo foram apresentados conceitos da tecnologia de banco de dados com o objetivo de familiarizar o leitor com o restante do trabalho apresentado nesta dissertação.

No restante deste trabalho serão apresentados conceitos da integração de bancos de dados bem como formalismos e metodologias que tentam diagnosticar e resolver o problema da integração.

Por fim será apresentada um sugestão de projeto de sistema de software para auxiliar o processo de integração através da automatização parcial.

CAPÍTULO 3

INTEGRAÇÃO DE BANCOS DE DADOS

Um dos maiores objetivos da integração de dados é fornecer uma interface integrada e uniforme para consultas em diversas fontes de dados que descrevam os mesmos objetos do mundo real. Assim um sistema de integração de dados possibilita ao usuário se concentrar no que ele deseja sem ter de pensar em como conseguir essa informação. O usuário se vê livre da tarefa de descobrir quais fontes de dados contém as informações relevantes, fazer o acesso e combinar os resultados em uma única resposta.

Existem muitas situações que utilizam as tecnologias de integração de bancos de dados. Um bom exemplo são organizações que a medida que evoluem precisam aumentar a sua estrutura criando novas unidades, cada uma delas com funções e localizações distintas. Normalmente essas novas unidades precisarão de uma parte ou até mesmo de todos os dados de cada uma das unidades já existentes.

Por esse motivo a interoperabilidade entre os dados dessas fontes se torna cada vez mais crucial para a obtenção de informações relevantes. Existem vários graus de interoperabilidade, desde gateways que permitem a comunicação de um par específico de DBMS (sistemas gerenciadores de bancos de dados) até sistemas de softwares que criam visões persistentes sobre várias fontes de dados sem se importar em conciliar as restrições já existentes sobre os dados de cada uma das fontes. Mas a interoperabilidade completa só pode ser alcançada por bancos de dados distribuídos ou federados porque eles permitem a união dos dados em um único banco de dados virtual.

Um banco de dados distribuído possui seus dados espalhados por vários locais em uma rede (muitas vezes até mesmo replicados em mais de um local da rede) de modo a facilitar seu acesso. Já os bancos de dados federados denotam um conjunto de fontes de dados locais que compartilham suas informações de modo explícito através da exportação de esquemas. Esses esquemas definem o que deve ser compartilhado de cada fonte. O que

difere sistemas de bancos de dados federados de sistemas de bancos de dados distribuídos é que no primeiro as fontes de dados foram criadas independentemente umas das outras e continuam funcionando de maneira autônoma.

De um modo mais formal, a integração de bancos de dados é o processo que toma como entrada um conjunto de bancos de dados e produz como saída uma única descrição dos esquemas dos bancos de dados de entrada (esquema integrado) e um mapeamento das informações associadas que permita o acesso as informações existentes nos bancos de dados de entrada através do esquema integrado.

3.1 Fases da Integração de Bancos de Dados

O estudo do processo de integração de bancos de dados nas suas diversas formas de abordagem e realização foi feito, entre outros, por C. Parent e S. Spaccapietra em [18] e por C. Batini, M. Lenzerini e S. B. Navathe em [2]. Eles dividiram o processo em fases conforme suas necessidades ou preocupações sugerindo as técnicas mais utilizadas para a sua resolução. Assim nem todas as fases que aparecem aqui foram definidas pelo mesmo autor ou pelo mesmo trabalho. Na verdade foi feita uma compilação dos trabalhos pelo autor desta dissertação de modo a se obter uma nova e mais completa divisão das fases da integração de bancos de dados. Essas fases são:

- considerações iniciais - fase onde os esquemas a serem integrados são transformados de modo a se tornarem sintaticamente e semânticamente mais homogêneos;
- pré-integração - consiste de uma análise dos esquemas a serem integrados com o objetivo de escolher uma política de integração eficiente, definindo, entre outras coisas, quais esquemas serão integrados e em que ordem ou se serão integrados apenas pedaços dos esquemas ao invés do esquema inteiro;
- identificação de correspondências - fase onde as correspondências entre os esquemas são identificadas e descritas;
- identificação de conflitos - fase onde os esquemas são analizados e comparados para

determinar as correspondências entre conceitos e detectar possíveis conflitos (conflitos de tipo, estrutura, etc.);

- integração - fase onde os conflitos entre os esquemas são resolvidos e os itens correspondentes são unificados;

- união e reestruturação - nesta fase são gerados esquemas integrados intermediários, para análise e se necessário, uma reestruturação para que se alcance qualidades desejáveis, como:

  - completude e corretude - a base de dados integrada precisa apresentar de forma correta todas as informações que estão nas fontes de dados locais;
  - minimalidade - quando uma mesma informação for representada em mais de uma das fontes de dados locais, deverá aparecer apenas uma vez na base de dados integrada;
  - entendibilidade - o esquema integrado deverá ser facil de entender, tanto para o administrador quanto para o usuário final.

3.1.1 Considerações Iniciais

Um pré-requisito para a integração de bancos de dados é uma visualização comum do funcionamento de todas as fontes de dados a serem integradas, tornando-as o mais homogêneas possível para comparações futuras.

Uma vez que pesquisadores da área de integração de banco de dados assumem que todos os bancos de dados utilizam o mesmo *modelo de dados comum (common data model - CDM)*, o trabalho de tradução se torna um pré-requisito da integração e é tratado como um problema a parte. Infelizmente a quantidade de soluções que façam a tradução de um CDM para outro automáticamente ainda é muito escassa.

Um problema que gera muita polêmica é a escolha do CDM para o esquema integrado. Muitos pesquisadores são favoráveis ao modelo orientado a objeto, uma vez que contém em si todos os conceitos dos outros modelos de dados e seus métodos podem ser usados

para implementar regras específicas de mapeamento. Em contrapartida, quanto mais rico é o modelo mais complexo se torna o processo de integração porque mais discrepâncias irão surgir devido as escolhas dos diferentes modelos de dados das fontes.

Para simplificar a integração deve-se escolher um CDM com o mínimo de semântica, onde as estruturas de dados sejam trazidas a estruturas elementares sem alternativas de modelagem, como os modelos de relacionamentos binários.

Deve ficar claro que a modelagem dos dados não é um processo determinístico. O relativismo semântico (onde cada desenvolvedor aplica a sua perspectiva ao criar o esquema) é o preço a se pagar pela riqueza semântica do modelo de dados a disposição.

3.1.2 Pré-Integração

Para qualquer metodologia, mesmo que ela considere uma fase de pré-integração ou não, existe a necessidade de considerar a sequência e o agrupamento das fontes de dados de entrada (mesmo que implicitamente).

Inicialmente deve-se considerar a quantidade de fontes de dados que devem ser integradas. Espera-se que esse número seja maior que dois mas em alguns casos o número exato não é constante e existe a possibilidade de serem acrescentadas ou retiradas fontes de dados ao processo de integração.

A seguir deve-se escolher uma política de integração das fontes a ser seguida. A figura 3.1 mostra quatro possíveis estratégias de processamento de fontes de dados para integração. Cada uma das estratégias está representada como uma estrutura de grafos, uma árvore. Os vértices que são folhas representam os esquemas das fontes de dados e os vértices que não são folhas representam estados intermediários da integração. O vértice raiz representa o resultado final da integração.

As estratégias binárias permitem que os esquemas sejam integrados aos pares. A estratégia em escada permite que a cada passo do processo de integração sejam integrado um esquema de uma fonte de dados com um estado intermediário. A estratégia balanceada requer que se divida as fontes de dados em pares para que a integração seja feita de modo simétrico. As estratégias N-Árias permitem a integração de vários esquemas de uma vez.

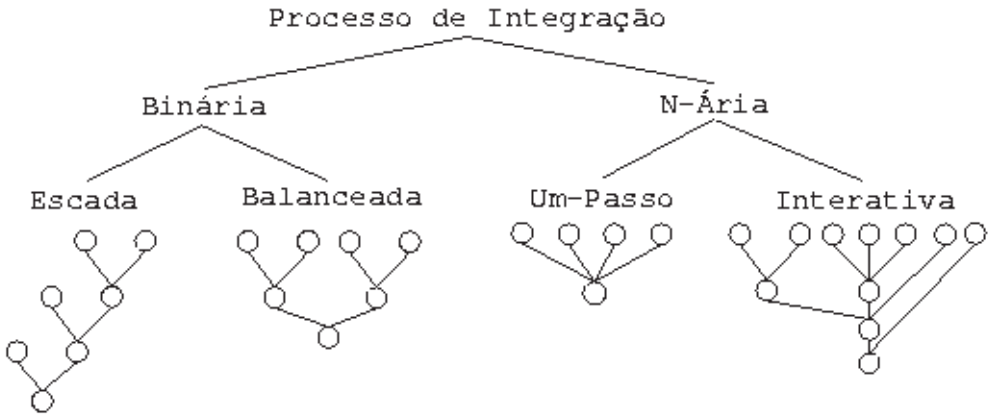

Figura 3.1: Estratégias de Processamento de Fontes de Dados

A estratégia em um-passo é a que tem o conceito mais simples e consiste de integrar todas as fontes de dados juntas e ao mesmo tempo. A estratégia interativa é aquela que segue qualquer outra política de integração que não tenha sido apresentada.

A maior vantagem das estratégias binárias está em simplificar as atividades de comparação entre esquemas dividindo-as em vários passos. Cada um desses passos analisa apenas dois esquemas de cada vez. Pesquisas indicam que em geral a complexidade da comparação entre n esquemas tem um grau de complexidade de n^2. Esse fato faz com que as estratégias binárias sejam as preferidas entre as estratégias utilizadas. Mas sua maior desvantagem está em gerar um grande número de passos para um número alto de fontes de dados.

As vantagens e desvantagens das estratégias N-Árias são diretamente opostas as vantagens e desvantagens das estratégias binárias.

3.1.3 Identificação de Correspondências

Um banco de dados contém representações de objetos do mundo real. Na fase de identificação de correspondências a integração de bancos de dados vai além das representações de modo a observar o que é representado ao invés de como é representado.

Dois bancos de dados possuem alguma coisa em comum se os objetos do mundo real que eles representam possuem elementos em comum ou são de algum modo relacionados. Dois objetos (instâncias ou valores) em dois bancos de dados correspondem um ao outro

se eles representam o mesmo objeto do mundo real.

Para que esse processo fique mais claro pode-se apresentar a metodologia mostrada em [18] onde as correspondências entre dois bancos de dados são representadas através de *afirmações de correspondências entre esquemas (interschema correspondence assertion - ICA)*. Um ICA se baseia nos tipos ao invés das instâncias para definir correspondências. De um modo extremamente simples pode-se dizer que um ICA seria uma descrição de quando e como dois objetos são equivalentes.

Para se definir completamente um ICA deve-se obedecer quatro passos.

1. Identificar os conjuntos de elementos em cada fonte de dados que representam o mesmo objeto do mundo real.

2. Definir que tipo de relacionamento entre conjuntos se mantém entre as extensões de cada uma das fontes de dados: equivalência ($\equiv$), inclusão ($\supseteq$), interseção ($\cap$) ou disjunção ($\neq$).

3. O sistema precisa saber como encontrar em um banco de dados o objeto (instância ou valor) correspondente a um determinado objeto em outro banco de dados. Portanto cada ICA precisa definir um mapeamento entre as instâncias correspondentes em meio a diversas fontes de dados.

4. Por fim é necessário identificar atributos que estejam em ambos os conjuntos de elementos de cada banco de dados com o objetivo de evitar duplicatas no esquema integrado.

Infelizmente, uma vez que se trabalhe com esquema grandes e reais, o processo de identificar todos os ICAs necessários para a integração não é nem simples nem trivial.

3.1.4 Identificação de Conflitos

A atividade fundamental dessa fase é a de identificar todos os conflitos existentes na representação de objetos correspondentes em esquemas diferentes.

Os casos de conflito existentes podem ser divididos em conflitos de incompatibilidade de domínio e conflitos entre definições de entidades, sendo melhor apresentados na seção 3.2.

3.1.5 Integração

Uma vez que todas as correspondências tenham sido descritas a integração pode começar. Cada correspondência é analizada de modo a decidir qual representação dos elementos relacionados deve ser escolhida para ser incluida no esquema integrado e qual esquema de mapeamento deve ser definido entre o esquema integrado e as fontes de dados. A representação escolhida deverá se ajustar as necessidades apresentadas permitindo a resolução de quaisquer conflitos de dados que se apresentem e a solução deverá ser acrescentada ao esquema de mapeamento.

Algumas vezes conflitos não podem ser resolvidos porque surgem como resultado de alguma discrepância básica entre os esquemas e nesse caso a melhor solução é o diálogo com os usuários e administradores para buscar um meio de contornar o problema.

A fase de integração pode ser realizada manualmente pelo administrador do banco de dados *(database administrator - DBA)* federado através de uma linguagem de manipulação procedimental, declarativa ou lógica. Essa fase também pode ser realizada semi-automaticamente por uma ferramenta (como por exemplo a ferramenta descrita em [23]). Nesse caso o DBA federado é responsável apenas pelas correspondências e pela escolha de algumas alternativas de integração.

Quando uma correspondência descreve dois objetos como idênticos (mesma representação e extensões equivalentes) a sua integração é simples é direta. Mas muitas vezes existem diferenças nas extensões ou na representação das mesmas, os assim chamados conflitos.

3.1.6 União e Reestruturação

Durante esta fase os mais variados tipos de operações são empregados tanto nos esquemas das fontes de dados como nos esquemas integrados para obter qualidades desejáveis

como completude, corretude, minimalidade e entendibilidade. Em algumas metodologias essa fase já possui um protótipo do banco de dados integrado para avaliação.

Para alcançar a completude e a corretude deve-se analisar o esquema integrado para averiguar se todos os dados que estão nas fontes de dados estarão presentes no banco de dados integrado de modo correto. A completude é rejeitada em muitos casos devido ao fato de que os desenvolvedores desejam que apenas determinado subconjunto dos dados relevante ao objetivo da integração esteja presente no banco de dados integrado.

O objetivo da minimalidade é o de descobrir e eliminar redundâncias que venham a ocorrer. Se uma informação está presente em mais de uma fonte de dados ela deverá aparecer apenas uma vez no banco de dados integrado.

A entendibilidade do esquema integrado é necessária para que usuário e administradores possam utilizar mais facilmente o banco de dados integrado e para que futuras manutenções e atualizações possam ser realizadas. Infelizmente não existem meios eficientes de se medir o quanto um esquema é entendível ou não, de modo que na maior parte dos casos a intuição termina por ser a melhor solução.

3.2 Casos de Conflito

Pessoas diferentes vêem os objetos do mundo real de formas diferentes. Portanto é justo que usem nomes, estruturas e hierarquias diferentes uns dos outros para representar estes objetos do mundo real. Essa situação pode gerar diversos conflitos para a integração de dados.

Classificar os tipos de conflitos que podem surgir entre duas representações de um mesmo objeto não é uma tarefa simples. Definir uma metodologia que os solucione é uma tarefa ainda mais difícil. A classificação e identificação dos casos de conflito relevantes na integração de banco de dados apresentada aqui foi inspirada nos trabalhos de Sheth e Kashyap [21] e W. Kim e J. Seo [8]. Técnicas para resolução destes casos de conflito pertinentes a metodologia empregada nesse trabalho são apresentadas na seção 4.6.

Para facilitar o entendimento serão apresentados alguns exemplos criados pelo autor desta dissertação para cada um dos casos de conflito. Os exemplos seguem modelo rela-

cional porque esse será o modelo de dados mais utilizado no restante do projeto. Cada caixa é uma relação. O nome da relação aparece no topo destacado em letras maiúsculas. Ao seu lado é exibido o esquema de origem da relação. Cada um dos nomes a seguir é um atributo da relação. Atributos sublinhados são atributos chaves da relação. O tipo do atributo aparece, quando relevante, ao lado do nome do atributo.

3.2.1 Conflitos de Incompatibilidade de Domínio

3.2.1.1 Conflitos de Nomes

Ocorre quando dois objetos semânticamente equivalentes possuem nomes diferentes ou quando dois objetos sem nenhuma equivalência semântica possuem nomes iguais. Uma vez que o problema tenha sido localizado, a resolução deste tipo de conflito é bem simples e parecida para os dois casos. Um exemplo deste caso é definido na tabela 3.1:

ESTUDANTE (esquema 1)
cpf
nome

ALUNO (esquema 2)
cpf
nome

Tabela 3.1: Exemplo de Conflito de Nomes

3.2.1.2 Conflitos de Representação dos Dados

Ocorre quando dois objetos semânticamente equivalentes são representados por diferentes tipos de dados. Um exemplo deste caso é definido na tabela 3.2:

ALUNO (esquema 1)
cpf
rg (string)
nome

ALUNO (esquema 2)
cpf
rg (inteiro)
nome

Tabela 3.2: Exemplo de Conflito de Representação dos dados

Neste exemplo o campo rg no esquema global é representado por uma string enquanto que no esquema 2 um campo com o mesmo nome é representado por um valor numérico.

3.2.1.3 Conflitos de Unidade

Ocorre quando dois objetos semânticamente equivalentes são representados em diferentes unidades ou medidas. Por exemplo, o tempo pode ser representado em anos em uma base e meses em outra base. Um exemplo deste caso é definido na tabela 3.3:

PACIENTE (esquema 1)
cpf
nome
pesoQ

PACIENTE (esquema 2)
cpf
nome
pesoL

Tabela 3.3: Exemplo de Conflito de Unidade

Neste exemplo, pesoQ está representado em quilos e pesoL está representado em libras.

3.2.1.4 Conflitos de Precisão dos Dados

Ocorre quando dois atributos que são semanticamente parecidos são representados com diferentes unidades e medidas de modo a não ocorrer uma correspondência de um para um entre os respectivos domínios. Um exemplo deste caso é definido na tabela 3.4:

ALUNO (esquema 1)
cpf
nome
conceito

ALUNO (esquema 2)
cpf
nome
nota

Tabela 3.4: Exemplo de Conflito de Precisão de Dados

Neste exemplo, o atributo conceito têm seus valores representados como A, B, C ou D. O atributo nota tem seus valores representados como um número inteiro de 0 a 100.

3.2.1.5 Conflitos de Valores Padrão

Este é um tipo muito peculiar de conflito. Ocorre quando em um atributo passa a ter um valor específico quando não é indicado um valor para ele. Um exemplo deste caso é definido na tabela 3.5:

PESSOAL (esquema 1)
cpf
nome
titulo

PESSOAL (esquema 2)
cpf
nome
titulovp

Tabela 3.5: Exemplo de Conflito de Valores Padrão

Neste exemplo, o atributo titulo tem seus valores representados por ESTUDANTE, PROFESSOR ou FUNCIONÁRIO. No esquema 2 o atributo titulovp funciona da mesma forma, mas quando seu valor for ESTUDANTE, a ocorrência será representada com o valor VOID (NULL).

3.2.2 Conflitos entre Definições de Entidades

3.2.2.1 Conflitos de Equivalência de Chaves

Ocorre quando duas entidades semanticamente parecidas são definidas com identificadores (chaves primárias) semanticamente diferentes. A resolução deste problema pode ser alcançada se for possível definir um mecanismo de mapeamento entre os identificadores. Nesse caso o problema geralmente se reduz a um conflito de unidades. Caso não seja possível encontrar uma função de mapeamento ou mesmo um identificador comum não será possível realizar a integração de modo satisfatório. Um exemplo deste caso é definido na tabela 3.6:

ALUNO (esquema 1)
cpf
rg
nome

ALUNO (esquema 2)
rg
cpf
nome

Tabela 3.6: Exemplo de Conflito de Equivalência de Chaves

Este é um caso simples, onde o identificador do esquema global é um atributo no esquema local e vice-versa, onde a função de mapeamento pode ser facilmente obtida. Existem muitos casos onde a solução não pode ser alcançada tão simplesmente.

3.2.2.2 Conflitos de União

Ocorre quando dois objetos semânticamente equivalentes são representados com um número diferente de atributos ou com atributos não relacionados de modo que a união entre eles não é compatível. Um exemplo deste caso é definido na tabela 3.7:

ALUNO (esquema 1)
cpf
nome
rg

ALUNO (esquema 2)
cpf
nome
altura

Tabela 3.7: Exemplo de Conflito de União

Neste exemplo, o atributo rg existe no esquema 1 mas nunca existiu no esquema 2. Já o atributo altura existe no esquema 2 mas não existe no esquema 1.

3.2.2.3 Conflitos de Isomorfismo

Ocorre quando um número diferente de atributos é utilizado para representar conceitos similares. Deve-se então definir uma função de mapeamento entre os atributos de um esquema e os atributos de outro. Um exemplo deste caso é definido na tabela 3.8:

ALUNO (esquema 1)
cpf
nomecompleto

ALUNO (esquema 2)
cpf
nome
sobrenome

Tabela 3.8: Exemplo de Conflito de Isomorfismo

Neste exemplo, o esquema 1 guarda o nome dos aluno no atributo nomecompleto enquanto que no esquema 2 o mesmo conceito é armazenado nos atributos nome e sobrenome.

3.2.2.4 Conflitos de Falta de Atributos

A maior parte dos casos onde isso pode ocorrer já foi mostrado neste capítulo. Mas existe um caso em especial onde o atributo que falta pode ser recuperado através de

mecanismos de inferência. Um exemplo deste caso é definido na tabela 3.9:

ALUNO (esquema 1)
cpf
nome
tipo

ALUNOGRAD (esquema 2)
cpf
nome

Tabela 3.9: Exemplo de Conflito de Falta de Dados

Neste exemplo, o esquema 1 guarda alunos de todos os tipos (graduação, pós-graduação, etc.). Já o esquema 2 guarda apenas os alunos de graduação. Apesar do atributo tipo não estar presente no esquema local, ele pode ser facilmente deduzido através de inferência uma vez que todos os alunos obtidos do banco de dados representado pelo esquema 2 serão alunos da graduação.

3.2.2.5 Conflitos entre Relações e Atributos

Ocorre quando o mesmo objeto é modelado como um atributo em um esquema e como uma relação em outro esquema. Um exemplo deste caso é definido na tabela 3.10:

ALUNO (esquema 1)
cpf
nome
idade
orientador

ALUNO (esquema 2)
cpf
nome
idade

ALUNOPG (esquema 2)
cpf
orientador

Tabela 3.10: Exemplo de Conflito entre Relação e Atributo

Neste exemplo, o esquema 1 guarda todas as informações sobre todos os alunos na mesma tabela. Já o esquema 2 guarda as informações gerais sobre os alunos em uma tabela e informações específicas sobre alunos da pós-graduação em outra tabela.

3.2.2.6 Conflitos entre Atributos e Dados

Ocorre quando o valor de um atributo em um esquema corresponde ao próprio atributo em outro esquema. Um exemplo deste caso é definido na tabela 3.11:

ALUNO (esquema 1)
cpf
nome
tipo

ALUNO (esquema 2)
cpf
nome
grad
mestre
doutor

Tabela 3.11: Exemplo de Conflito entre Atributo e Dados

Neste exemplo, o esquema 1 guarda no atributo tipo a condição do aluno (grad, mestre, doutor). Já o esquema 2 possui um atributo do tipo booleano para cada condição possível.

3.3 Formalismos para Integração de Bancos de Dados

Uma das principais diferenças entre um sistema de integração de dados e um sistema de banco de dados tradicional é o fato de que as consultas são submetidas pelos usuários a um esquema global, mesmo que os dados estejam sendo armazenados em esquemas locais. Então, de modo a permitir que o sistema de integração de banco de dados possa responder as consultas, deve haver alguma descrição do relacionamento entre o esquema global e os esquemas locais. O processador de consultas deve estar apto a reformular uma consulta submetida a ele como novas consultas submetidas aos esquemas locais [9].

Em princípio, poderia-se usar fórmulas lógicas de primeira ordem para descrever as fontes de dados. Mas nesse caso a reformulação completa seria praticamente impossível. Então várias aproximações foram desenvolvidas nas quais formas restritas de fórmulas lógicas de primeira ordem acompanhadas de algorítimos de reformulação têm sido utilizadas. Serão descritas duas dessas aproximações, "Global as View"(GAV) e "Local as View"(LAV).

3.3.1 Global as View (GAV)

Para cada relação pertencente ao esquema global, escreve-se uma consulta sobre os esquemas locais especificando como obter as instâncias destinadas a essa relação.

Reformulação de consultas em GAV é algo relativamente direto. Uma vez que as relações no esquema global estão definidas em termos das relações locais, tudo o que se

tem a fazer é abrir as definições das relações do esquema global. Entretanto, adicionar novas fontes ao sistema de integração não é um processo trivial. Para uma nova fonte de dados, é necessário descobrir todos os modos pelos quais se possa obter dados para cada uma das relações do esquema global.

O formalismo GAV é provavelmente o mais largamente utilizado e pesquisado. Maiores informações sobre ele podem ser acessadas no projeto TSIMMIS [6] onde diversas ferramentas para integração de fontes de dados heterogêneas têm sido desenvolvidas.

3.3.2 Local as View (LAV)

Nesta técnica cada uma das relações das fontes é descrita através de consultas sobre o esquema global.

Reformulação de consultas em LAV é uma operação mais complexa do que em GAV porque não é possível simplesmente abrir as definições das relações do esquema global. Algorítimos para a resolução desse problema são fornecidos em [9]. Em compensação é fácil adicionar novas fontes de dados, uma vez que se terá de escrever consultas apenas para cada uma das relações da nova fonte.

Um bom exemplo de metodologia que utiliza esse formalismo pode ser encontrado em [10] onde cada uma das fontes de dados é estudada de forma a se obter uma descrição declarativa do seu conteúdo e capacidade para resolução de consultas. Essa metodologia é considerada centrada nas fontes de dados e não nas consultas porque utiliza as descrições das fontes para gerar os planos de resposta das consultas.

CAPÍTULO 4

O FORMALISMO E SUAS METODOLOGIAS

4.1 O Projeto AutoMed

O *Projeto de Geração Automática de Ferramentas Mediadoras para Integração de Bancos de Dados Heterogêneos (Automatic Generation of Mediator Tools for Heterogeneous Database Integration - AutoMed)* [3] é uma parceria de pesquisas realizada pela *Universidade de Birkbeck (Birkbeck University of London)* e pela *Universidade Imperial (Imperial College of London)*.

A metodologia utilizada pelo projeto AutoMed se caracteriza por ver o processo de integração como uma sequência de transformações reversíveis que modificam tanto o esquema como a extensão do banco de dados. Essa sequência de transformações incrementalmente adiciona, apaga ou renomeia um objeto do esquema do banco de dados até que se possa mapear diversos esquemas de bancos de dados uns com os outros. Acompanhando cada uma dessas transformações há uma consulta que especifica como as instâncias do objeto que está sendo adicionado ou apagado podem ser obtidas a partir de outros objetos do esquema.

Talvez a maior vantagem do projeto AutoMed seja a de que ele não está restrito a apenas uma linguagem de modelagem para bancos de dados. Ao invés disso as linguagens de modelagem podem ser representadas através de um modelo de dados baseado em hipergrafos (seção 4.2). Em consequência disso basta configurar a linguagem de modelagem escolhida em termos do modelo de dados baseado em hipergrafos para que seja possível utilizar a implementação existente do projeto.

A implementação do projeto AutoMed já possui um repositório de dados que funciona através de Java API e provê um armazenamento persistente das informações de configurações da linguagem de modelagem dos dados, dos esquemas dos bancos de dados e das sequências de transformação entre esses esquemas. Maiores informações sobre o

funcionamento do repositório de dados com relação a representação das linguagens de modelagem e sua integração podem ser encontradas em [4].

4.2 O Modelo de Dados Baseado em Hipergrafos

Muitos formalismos e metodologias já foram criados para resolver o problema da integração de bancos de dados. A. Poulovassilis e P. McBrien criaram um formalismo [12] que procura eliminar as diferenças entre esquemas através de transformação dos mesmos. Este formalismo é denominado *modelo de dados baseado em hipergrafos (hipergraph data model - HDM).*

Este formalismo utiliza o conceito de hipergrafos para representar o esquema, as instâncias e o mapeamento entre ambos, representando assim um modelo de dados.

Para trabalhar com esse formalismo foram definidas transformações primitivas que são listadas a seguir:

- $renameNode(fromName, toName)$ - Renomeia um vértice.
- $renameEdge(\langle fromName, c_1, \ldots, c_m \rangle, toName)$ - Renomeia uma aresta.
- $addConstraint\ c$ - Adiciona uma nova restrição c.
- $delConstraint\ c$ - Apaga uma restrição c.
- $addNode(name, q)$ - Adiciona um vértice chamado *name* cuja extensão é dada pela consulta q.
- $delNode(name, q)$ - Apaga um vértice chamado *name*. q é um consulta que determina como a extensão do vértice apagado pode ser recuperada a partir dos objetos restantes do esquema.
- $addEdge(\langle name, c_1, \ldots, c_m \rangle, q)$ - Adiciona uma nova aresta entre uma sequência de objetos $c_1, \ldots, c_m$ do esquema. A extensão da aresta é determinada pelo valor da consulta q.

- $delEdge(\langle name, c_1, \ldots, c_m \rangle, q)$ - Apaga uma aresta. q é um consulta que determina como a extensão da aresta apagada pode ser recuperada a partir dos objetos restantes do esquema.

Aqueles que desejarem se aprofundar mais nesse formalismo podem consultar o anexo A onde encontrarão uma especificação bem mais completa. Mais informações sobre a utilização e evolução deste formalismo podem ser encontradas em [15] onde foi demonstrado que o formalismo consegue suportar todas as trasnformações usadas na integração de esquemas utilizando o modelo de dados entidade-relacionamento e em [17] onde foi demonstrado que o formalismo pode ser aplicado para várias linguagens de modelagem de alto nível como o modelo de dados entidade-relacionamento, relacional, documentos da Internet e até mesmo UML.

4.3 Linguagem Intermediária para Consultas (IQL)

Como dito anteriormente, o HDM não possui uma linguagem própria para consultas e restrições. Ele permite que o desenvolvedor escolha a linguagem utilizada para dar mais liberdade ao formalismo. Como o formalismo BAV (seção 4.4) usa uma linguagem específica então ela será apresentada através de um breve resumo. Não é necessário aprofundar o conhecimento da linguagem aqui uma vez que este trabalho não utiliza todo o seu potencial.

A *Linguagem Intermediária para Consultas (Intermediate Query Language - IQL)* [19, 7, 20] é uma linguagem que pertence ao paradigma funcional. Seu propósito é ser uma linguagem para consultas com fácil tradução para várias linguagens de alto nível, como SQL por exemplo.

A linguagem suporta strings de caracteres, valores booleanos, números reais e números inteiros. Ela trabalha com vários tipos de coleções como conjuntos (representados na forma $\{1, 2, 3\}$), multi-conjuntos (conjuntos que aceitam repetições representados na forma $\{|1, 2, 3|\}$), listas (representadas na forma $[1, 2, 3]$) e tuplas (representadas na forma $(1, 2, 3)$).

Várias funções nativas são definidas pela linguagem. Estão entre elas: $(+)$, $(-)$, $(*)$, $(/)$, $(=$

), $(!=)$, $(<)$, $(>)$, $(<=)$, $(>=)$, and, or, not, if. As funções costumam ser utilizadas na forma prefixa com exceção dos operadores $(++)$ e $(--)$ que representam respectivamente a união e diferença entre coleções e podem ser utilizados tanto na forma prefixa quanto na forma infixa.

Funções anônimas podem ser definidas através de *abstrações lambda*. O exemplo a seguir define uma função que retorna o valor de uma expressão que recebe três argumentos, soma os dois primeiros e multiplica pelo terceiro.

$$\lambda\ (x,\ y,\ z)\ ((*)\ ((+)\ x\ y)\ z)$$

Expressões do tipo *let* também são aceitas pela linguagem. Sua utilidade está em ligar uma expressão a um nome. Desse modo pode-se tornar menor uma expressão que use o mesmo código repetidamente. O exemplo a seguir repete o exemplo anterior por duas vezes somando seus resultados e retornado o valor 44.

$$let\ v = \lambda\ (x,\ y,\ z)\ ((*)\ ((+)\ x\ y)\ z)\ in\ ((+)\ (v(1,\ 2,\ 3))\ (v(3,\ 4,\ 5)))$$

Mas o que torna a linguagem agradável e fácil de usar é a utilização da *sintaxe de compreensão* [5]. O uso da sintaxe de compreensão não gera nenhuma expressividade adicional a linguagem mas torna a linguagem sintaticamente fácil de ler e entender. Além disso é esperado que seja uma tarefa fácil a tradução do IQL para outras linguagens de alto nível e vice-versa.

Uma compreensão se apresenta na forma $[e\ |Q_1; \ldots ; Q_n]$ para $n \geq 0$. Ela é composta por uma expressão inicial que denota o formato do resultado seguido por uma sequência de qualificadores que podem ser geradores ou filtros. Geradores relacionam uma variável ou uma tupla de variáveis com uma expressão que denota uma lista de valores. Filtros são expressões booleanas que selecionam as instâncias devolvidas pelos geradores.

Assumindo que R e S sejam coleções, o conjunto das operações para manipulação de dados no modelo relacional podem ser utilizados da seguinte maneira através da linguagem:

- União: $R + +S$ ou $(++)R\ S$

- Diferença: $R - -S$ ou $(--)R\ S$
- Projeção: $[(x, z) \mid (x, y, z) \leftarrow R]$
- Seleção: $[(x, y, z) \mid (x, y, z) \leftarrow R; (>)\ z\ 10]$
- Produto Cartesiano: $[(x_1, y_1, x_2, y_2) \mid (x_1, y_1) \leftarrow R; (x_2, y_2) \leftarrow S]$

A linguagem intermediária para consultas está se tornando rapidamente a principal linguagem do projeto AutoMed sendo utilizada em grande escala para representar as operações de transformação de seus formalismos e para representar as consultas disparadas contra a sua implementação de um repositório de dados.

4.3.1 Sintaxe

```
query   ::=  expr
         |   Let VarToken Equal query In query
         |   query Append query
         |   query Difference query
         |   query expr

expr    ::=  NumToken
         |   StrToken
         |   VarToken
         |   Any
         |   BoolToken
         |   Void
         |   scheme
         |   VarToken Colon scheme
         |   LSB query Bar quals RSB
         |   LSB RSB
```

	|	LSB seq RSB
	|	LCB seq RCB
	|	LRB query RRB
	|	Lambda pattern expr
seq	::=	seq Comma query
	|	query
quals	::=	qual SemiColon quals
	|	qual
qual	::=	query
	|	pattern LArrow query
pattern	::=	VarToken
	|	LCB pattern_seq RCB
pattern_seq	::=	pattern_seq Comma pattern
	|	pattern
scheme	::=	LDAB scheme_seq RDAB
scheme_seq	::=	scheme_element
	|	scheme_seq Comma scheme_element
scheme_element	::=	VarToken
	|	StrToken
	|	NumToken
	|	scheme

BoolToken = "True"
| "False"

StrToken = \' [∧\']* \'

NumToken = $[0-9]^+$
| $([0-9]^+)(.)([0-9]^+)$

VarToken = "(+)" | "(−)" | "(∗)"
| "(/)" | "(&)" | "(#)"
| "(=)" | "(! =)" | "(<)"
| "(>)" | "(<=)" | "(>=)"
| $[A-Za-z\_]\ [A-Za-z0-9\_\$]^*$

Let = "let"

In = "in"

Equal = " = "

Append = " + +"

Difference = "−−"

SemiColon = ";"

LArrow = "<−"

Comma = ","

Bar = " | "

LSB = "["

RSB = "]"

LRB = "("

RRB = ")"

LCB = "{"

RCB = "}"

Any	=	”any”
Lambda	=	”lambda”
LDAB	=	”<<”
RDAB	=	”>>”
Colon	=	” : ”
Void	=	”void”

4.3.2 Funções Nativas

Operadores Unários Prefixados

- not exp - devolve a negação de uma expressão;
- count list - devolve o comprimento de uma lista;
- sort list - devolve os elementos de uma lista;
- distinct list - retira os elementos duplicados de uma lista;
- group list - devolve uma lista de pares onde o primeiro elemento de cada par pertence a lista fornecida como parâmetro e o segundo componente é uma lista;
- max list - devolve o maior valor de uma lista de números;
- min list - devolve o menor valor de uma lista de números;
- sum list - devolve a soma dos valores de uma lista;
- avg list - devolve a média dos valores de uma lista.

Operadores Binários Infixos

- list ++ list - devolve a união de duas listas;
- list −− list - devolve a diferença entre duas listas.

Operadores Binários Prefixados

- operadores básicos - (+) (−) (∗) (/) (=) (! =) (>) (<) (>=) (<=) and or;
- setUnion list list - devolve a união de duas listas após retirar as duplicatas;
- sub list list - devolve verdadeiro se a primeira lista for um subconjunto da segunda;
- intersect list list - devolve a intersecção entre duas listas;
- member list exp - devolve verdadeiro se a expressão pertence a lista;
- map func list - aplica a função a cada um dos valores da lista;
- flatmap func list - aplica uma função que devolve uma lista a cada um dos elementos de uma lista devolvendo a concatenação das listas resultantes.

Operador Ternário

- if exp exp exp - se o valor da primeira expressão for verdadeiro devolve o valor da segunda expressão, caso contrário devolve o valor da terceira expressão.

4.4 Both as View (BAV)

O formalismo *both as view (BAV)* [13, 14, 11] foi desenvolvido a partir do formalismo HDM e dos formalismos GAV e LAV e desenvolvido de modo a ser aplicado em esquemas que usem o modelo de dados relacional. Projeções do formalismo para o modelo de dados entidade relacionamento, diagramas UML e documentos da Internet podem ser encontrados em [17]. Diante dessas projeções avalia-se que será uma tarefa simples estender a implementação do repositório de dados do projeto AutoMed de modo a ser aplicado a outros modelos de dados.

Cada objeto pertencente a um esquema de um banco de dados é representado através de um *scheme* pertencente ao formalismo HDM (ver seção 4.2). Assim cada objeto de um banco de dados relacional é representado da seguinte maneira:

- Uma relação é representada através da forma $<< rel >>$ onde *rel* é o nome da relação. Suas instâncias representam o conjunto de chaves primárias da relação.

- Um atributo é representado através da forma $<< rel, att, const >>$ onde *rel* representa o nome da relação ao qual o atributo pertence, *att* representa o nome do atributo e *const* é uma restrição opcional que expressa se o valor do atributo pode ser nulo ou não. Suas instâncias representam o conjunto de chaves primárias da relação e em sequência o valor do atributo.

- Uma restrição por chaves primárias é representada através da forma $<< rel, att_1, \ldots, att_n >>$ onde *rel* representa o nome da relação ao qual a restrição pertence e $att_1, \ldots, att_n$ representa a lista de atributos pertencentes a restrição. Uma vez que é apena uma restrição ela não toma instâncias para si.

- Uma restrição por chaves estrangeiras é representada através da forma $<< rel, a_1, \ldots, a_n, rel_e, ea_1, \ldots, ea_n >>$ onde *rel* representa o nome da relação, rel_e representa a relação estrangeira, $a_1, \ldots, a_n$ representa uma lista de atributos de *rel* e $ea_1, \ldots, ea_n$ representa uma lista de chaves primárias de rel_e. Para que a restrição seja satisfeita $a_1, \ldots, a_n$ devem ser iguais a $ea_1, \ldots, ea_n$.

Uma visualização das estruturas que representam instâncias em um grafo é mostrada na figura 4.1. Deve-se observar que existe uma lista de chaves primárias e uma lista de atributos. Como chaves primárias também são atributos isso gera alguma redundância visto que elas são representadas duas vezes. Isto foi feito porque a uniformidade no tratamento dos atributos aumenta a facilidade com que consultas IQL são traduzidas em consultas SQL além de facilitar a codificação e implementação do repositório de dados do Projeto Automed.

Do mesmo modo que o formalismo HDM, o formalimo BAV se caracteriza por uma sequência de transformações aplicadas em sequência onde cada uma adiciona, retira ou renomeia um objeto do formalismo. Além disso são especificadas operações de *extend* e *contract* que respectivamente adicionam ou retiram objetos do formalismo especificando a consulta que determina a origem das instâncias apenas parcialmente ou até mesmo não as especificando. As possíveis transformaços do formalismo são:

- $addRel(<< rel >>, q)$ - Adiciona uma relação chamada *rel* ao esquema. A consulta

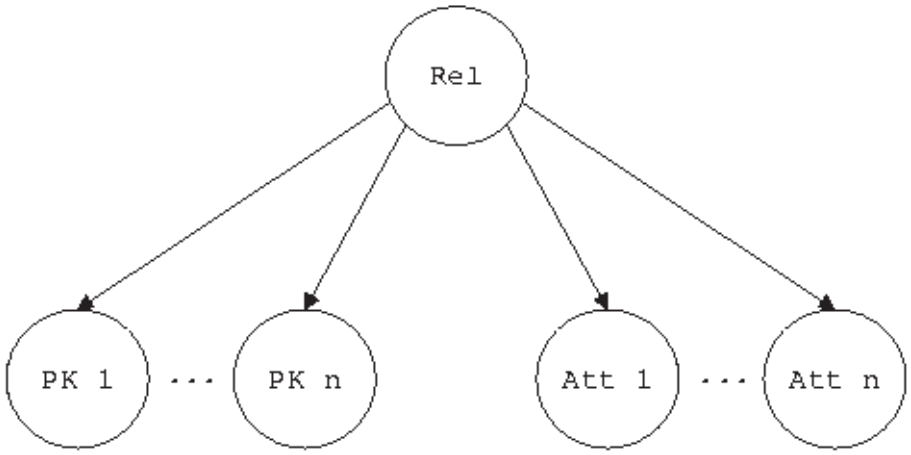

Figura 4.1: Grafo do Modelo Relacional em BAV

q especifica como é possível obter o conjunto de chaves primárias a partir dos objetos já existentes no esquema.

- $extRel(<< rel >>, q)$ - Adiciona uma relação chamada *rel* ao esquema. A consulta q é opcional, podendo ser substituída pela constante *void*, e especifica parcialmente como é possível obter o conjunto de chaves primárias a partir dos objetos já existentes no esquema.

- $delRel(<< rel >>, q)$ - Remove uma relação chamada *rel* do esquema. A consulta q especifica como é possível recuperar o conjunto de chaves primárias da relação excluida a partir dos objetos já existentes no esquema.

- $conRel(<< rel >>, q)$ - Remove uma relação chamada *rel* do esquema. A consulta q é opcional, podendo ser substituída pela constante *void*, e especifica como é possível recuperar o conjunto de chaves primárias da relação excluída a partir dos objetos já existentes no esquema.

- $renRel(<< rel_{old} >>, << rel_{new} >>)$ - Renomeia uma relação substituindo rel_{old} por rel_{new}.

- $addAtt(<< rel, att >>, q)$ - Adiciona um atributo chamado *att* na relação *rel*. A consulta q especifica como é possível obter o conjunto de instâncias do atributo a partir dos objetos já existentes no esquema.

- $extAtt(<< rel, att >>, q)$ - Adiciona um atributo chamado *att* na relação *rel*. A consulta q é opcional, podendo ser substituída pela constante *void*, e especifica parcialmente como é possível obter o conjunto de instâncias do atrIbuto a partir dos objetos já existentes no esquema.

- $delAtt(<< rel, att >>, q)$ - Remove um atributo chamado *att* da relação *rel*. A consulta q especifica como é possível recuperar o conjunto de instâncias do atributo excluído a partir dos objetos já existentes no esquema.

- $conAtt(<< rel >>, q)$ - Remove um atributo chamada *att* da relação *rel*. A consulta q é opcional, podendo ser substituída pela constante *void*, e especifica como é possível recuperar as instâncias do atributo excluído a partir dos objetos já existentes no esquema.

- $renAtt(<< rel, att_{old} >>, << rel, att_{new} >>)$ - Renomeia um atributo de uma relação *rel* substituindo att_{old} por att_{new}.

- $addPK(<< rel, a_1, \ldots, a_n >>)$ - Adiciona a chave primária da relação *rel*. A lista de atributos $a_1, \ldots, a_n$ indica os atributos que serão o conjunto de chaves primárias da relação.

- $delPK(<< rel, a_1, \ldots, a_n >>)$ - Remove a chave primária da relação *rel*. A lista de atributos $a_1, \ldots, a_n$ indica os atributos que eram o conjunto de chaves primárias da relação.

- $addFK(<< rel, a_1, \ldots, a_n, rel_e, ea_1, \ldots, ea_n >>)$ - Adiciona uma chave estrangeira entre a relação *rel* e a relação estrangeira rel_e. A lista de atributos $a_1, \ldots, a_n$ da relação *rel* deverá estar na lista de atributos chaves $ea_1, \ldots, ea_n$ da relação estrangeira rel_e para satisfazer a restrição.

- $delFK(<< rel, a_1, \ldots, a_n, rel_e, ea_1, \ldots, ea_n >>)$ - Remove uma chave estrangeira entre a relação *rel* e a relação estrangeira rel_e. A lista de atributos $a_1, \ldots, a_n$ da relação *rel* e os atributos chaves $ea_1, \ldots, ea_n$ da relação estrangeira rel_e devem ser indicados para realização da operação.

As transformação que adicionam ou removem atributos dos esquemas podem opcionalmente ter um parâmetro a mais indicando se o atributo pode ou não ter um valor nulo. A maioria dos trabalhos e implementações simplificam o formalismo *BAV* ignorando transformações que manipulem chaves primárias e chaves estrangeiras uma vez que não são necessárias em seus métodos.

Comparações entre o formalismo BAV e os formalismos GAV e LAV (seção 3.3) mostram algumas vantagens em relação aos mesmos: as evoluções nos esquemas locais e globais são mais bem aceitas pelo formalismo BAV uma vez que basta acrescentar novas transformações a sequência de transformações já existentes evitando a reconstrução do trabalho e o formalismo BAV também armazena mais informações sobre as extensões relativas a cada esquema, armazenando não só como recuperar as instâncias mas também se a recuperação será total ou parcial.

Além disso, a propriedade da reversibilidade das operações, herdada pelo formalismo HDM aplicada as visões GAV e LAV do esquemas aumenta a capacidade de resolução de consultas disparadas tanto às fontes de dados locais quanto ao banco de dados global.

4.5 Metodologias Utilizadas

O Projeto AutoMed possui diversas metodologias para realizar a integração de dados através do formalismo BAV. Todas elas se caracterizam por produzir sequências de transformações aplicadas tanto no esquema quanto nas instâncias de cada uma das fontes de dados. Duas metodologias se destacam por serem as mais utilizadas. Ambas se caracterizam por utilizar uma estratégia de integração de um único passo (seção 3.1.2).

A primeira metodologia [13] se destaca por uma postura mais artesanal, direcionada a casos específicos onde não se pretende adicionar ou retirar fontes de dados do processo de integração após seu término. Esta metodologia dá grande ênfase a completude do processo de integração, incentivando a idéia de que toda as informações das fontes de dados locais devem estar no banco de dados global. Costuma ser utilizada, por exemplo, em casos onde grandes corporações precisam integrar bancos de dados muito extensos.

Inicialmente, para cada objeto semântico pertencente as fontes de dados, adiciona-se

um objeto ao esquema do banco de dados global especificando de quais objetos das fontes de dados devem ser obtidas as respectivas instâncias. Uma vez que se tenha terminado de adicionar objetos ao esquema do banco de dados global deve-se retirar cada um dos objetos de cada uma das fontes de dados locais especificando como recuperar suas instâncias a partir do esquema do banco de dados global. A adição de objetos ao esquema do banco de dados global e a retirada dos objetos dos esquemas das fontes de dados locais geram satisfatóriamente os aspectos GAV e LAV do processo de integração.

A seguir serão mostrados os passos da integração de dois bancos de dados simples compostos por apenas uma relação cada um. Seus esquemas serão exibidos na tabela 4.1. A resolução de situações mais complexas através formalismo BAV será mostradas na seção 4.6.

ALUNO (esquema 1)
cpf
nome
ano

ESTUDANTE (esquema 2)
cpf
nome
orientador

ALUNO (esquema global)
cpf
nome
orientador
ano

Tabela 4.1: Esquemas para o Exemplo de Integração de Esquemas

As siglas esq_1, esq_2 e esq_g denotam respectivamente esquema 1, esquema 2 e esquema global. Nomes escritos com letras maiúsculas denotam nomes de relações e nomes escritos com letras minúsculas denotam nomes de atributos. Considera-se que todos os objetos dos esquemas locais já foram criados no formalismo. Portanto adiciona-se cada objeto semântico pertencente aos esquemas locais ao esquema global.

- $addRel(\langle\langle esq_g.ALUNO\rangle\rangle, [(x)|(x) \leftarrow \langle\langle esq_1.ALUNO\rangle\rangle] + +[(x)|(x) \leftarrow \langle\langle esq_2.ESTUDANTE\rangle\rangle])$
- $addAtt(\langle\langle esq_g.ALUNO, cpf\rangle\rangle, [(x,x)|(x) \leftarrow \langle\langle esq_g.ALUNO\rangle\rangle])$
- $addAtt(\langle\langle esq_g.ALUNO, nome\rangle\rangle, [(x,y)|(x,y) \leftarrow \langle\langle esq_1.ALUNO, nome\rangle\rangle; not\ (x) \leftarrow \langle\langle esq_2.ESTUDANTE\rangle\rangle] + +[(x,y)|(x,y) \leftarrow \langle\langle esq_2.ESTUDANTE, nome\rangle\rangle])$

- $addAtt(\langle\langle esq_g.ALUNO, orientador\rangle\rangle, [(x,y)|(x,y) \leftarrow \langle\langle esq_2.ESTUDANTE, orientador\rangle\rangle])$
- $addAtt(\langle\langle esq_g.ALUNO, ano\rangle\rangle, [(x,y)|(x,y) \leftarrow \langle\langle esq_1.ALUNO, ano\rangle\rangle])$
- $addPK(\langle\langle esq_g.ALUNO, \langle\langle esq_g.ALUNO, cpf\rangle\rangle\rangle\rangle)$

Resta agora retirar cada objeto pertencente aos esquemas locais.

- $delPK(\langle\langle esq_1.ALUNO, \langle\langle esq_1.ALUNO, cpf\rangle\rangle\rangle\rangle)$
- $delAtt(\langle\langle esq_1.ALUNO, ano\rangle\rangle, [(x,y)|(x,y) \leftarrow \langle\langle esq_g.ALUNO, ano\rangle\rangle;\ (!=)\ y\ void])$
- $delAtt(\langle\langle esq_1.ALUNO, nome\rangle\rangle, [(x,y)|(x,y) \leftarrow \langle\langle esq_g.ALUNO, nome\rangle\rangle; (x) \leftarrow \langle\langle esq_1.ALUNO\rangle\rangle;])$
- $delAtt(\langle\langle esq_1.ALUNO, cpf\rangle\rangle, [(x,x)|(x) \leftarrow \langle\langle esq_1.ALUNO\rangle\rangle)$
- $delRel(\langle\langle esq_1.ALUNO\rangle\rangle, [(x)|(x,y) \leftarrow \langle\langle esq_g.ALUNO, ano\rangle\rangle;\ (!=)\ y\ void])$
- $delPK(\langle\langle esq_2.ESTUDANTE, \langle\langle esq_2.ESTUDANTE, cpf\rangle\rangle\rangle\rangle)$
- $delAtt(\langle\langle esq_2.ESTUDANTE, orientador\rangle\rangle, [(x,y)|(x,y) \leftarrow \langle\langle esq_g.ALUNO, orientador\rangle\rangle;\ (!=)\ y\ void])$
- $delAtt(\langle\langle esq_2.ESTUDANTE, nome\rangle\rangle, [(x,y)|(x,y) \leftarrow \langle\langle esq_g.ALUNO, nome\rangle\rangle; (x) \leftarrow \langle\langle esq_2.ESTUDANTE\rangle\rangle])$
- $delAtt(\langle\langle esq_2.ESTUDANTE, cpf\rangle\rangle, [(x,x)|(x) \leftarrow \langle\langle esq_2.ESTUDANTE\rangle\rangle)$
- $delRel(\langle\langle esq_2.ESTUDANTE\rangle\rangle, [(x)|(x,y) \leftarrow \langle\langle esq_g.ALUNO, orientador\rangle\rangle; (!=)\ y\ void])$

A segunda metodologia [11] se destaca por uma postura de trabalho como a de uma linha de montagem. Por um lado é mais flexível, uma vez que é razoavelmente fácil adicionar ou retirar fontes de dados ao processo de integração mesmo após seu término. Por outro lado a completude do processo de integração é ignorada uma vez que normalmente

nem todas as informações das fontes de dados locais estarão presentes no banco de dados global. Costuma ser utilizada, por exemplo, em casos em que existe a necessidade de integrar muitos bancos de dados de tamanho reduzido como locadoras ou escolas.

O primeiro passo desta metodologia é definir um esquema que possa armazenar todas as informações desejadas no esquema do banco de dados global. Esse esquema, definido como *esquema de união*, é construído baseado nas aspirações do objetivo do banco de dados integrado. A partir daí, para cada fonte de dados local, deve-se definir uma sequência de transformações que torne seu esquema idêntico ao esquema de união. Uma vez que cada um dos esquemas das fontes de dados locais já foi redefinido com base no esquema de união basta criar uma rotina de transformações que faça a união de todos os esquemas compatíveis à união removendo as duplicatas e crie o esquema do banco de dados global de modo muito parecido com a metodologia anterior. A figura 4.2 consegue dar uma visão geral e mais clara de todos os passos dessa metodogia.

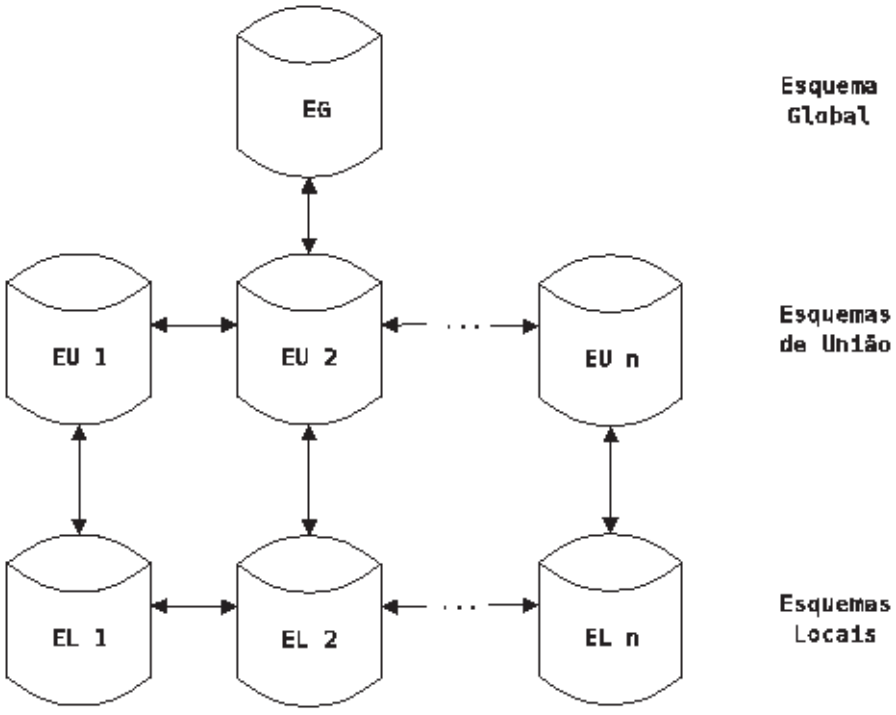

Figura 4.2: Processo de Integração que Utiliza Esquemas Compatíveis à União

A última etapa dessa metodologia, a junção de todos os esquemas compatíveis à união, já é realizada automaticamente pela implementação do Projeto AutoMed. Essa metodologia têm uma tendência maior a perder o aspecto LAV do processo de integração devido a

sua automatização, mas suas vantagens compensam isso.

4.6 Resolução dos Casos de Conflito

O objetivo deste capítulo é analisar cada um dos possíveis tipos de conflitos de dados que possam surgir durante o processo de integração e sua solução. Serão utilizados os conflitos apresentados na seção 3.2 através dos quais o autor desta dissertação irá demonstrar como o formalismo BAV pode solucionar cada conflito.

A metodologia do formalismo BAV utilizada aqui é baseada em um esquema de união. Assim um conflito estará solucionado quando o esquema local se tornar igual ao esquema de união. Em decorrência disso todas as transformações são aplicadas nos esquemas locais.

4.6.1 Conflitos de Incompatibilidade de Domínio

4.6.1.1 Conflitos de Nomes

ESTUDANTE (esquema de união)
cpf nome

ALUNO (esquema local)
cpf nome

Tabela 4.2: Exemplo de Conflito de Nomes

Neste exemplo as relações Estudante e Aluno possuem o mesmo significado semântico apesar de possuirem nomes diferentes. A resolução através do formalismo BAV seria feita assim:

- $renRel(\langle\langle ALUNO\rangle\rangle, \langle\langle ESTUDANTE\rangle\rangle)$

4.6.1.2 Conflitos de Representação dos Dados

Neste exemplo o campo rg no esquema de união é representado por uma string enquanto que no esquema local um campo com o mesmo nome é representado por um valor numérico. As funções int2str e str2int, responsáveis respectivamente por converter um número em string e uma string em número, não existem na definição da linguagem IQL, mas as

ALUNO (esquema de união)
cpf
rg (string)
nome

ALUNO (esquema local)
cpf
rg (inteiro)
nome

Tabela 4.3: Exemplo de Conflito de Representação dos dados

características da linguagem permitem que se defina novas funções. A resolução através do formalismo BAV seria feita assim:

- $renAtt(\langle\langle ALUNO, rg\rangle\rangle, \langle\langle ALUNO, rgn\rangle\rangle)$
- $addAtt(\langle\langle ALUNO, rg\rangle\rangle, [(x, int2str(y))|(x, y) \leftarrow \langle\langle ALUNO, rgn\rangle\rangle])$
- $delAtt(\langle\langle ALUNO, rgn\rangle\rangle, [(x, str2int(y))|(x, y) \leftarrow \langle\langle ALUNO, rg\rangle\rangle])$

4.6.1.3 Conflitos de Unidade

PACIENTE (esquema de união)
cpf
nome
pesoQ

PACIENTE (esquema local)
cpf
nome
pesoL

Tabela 4.4: Exemplo de Conflito de Unidade

Neste exemplo, pesoQ está representado em quilos e pesoL está representado em libras. A resolução através do formalismo BAV seria feita assim:

- $addAtt(\langle\langle PACIENTE, pesoQ\rangle\rangle, [(x, (y{*}0, 4536))|(x, y) \leftarrow \langle\langle PACIENTE, pesoL\rangle\rangle])$
- $delAtt(\langle\langle PACIENTE, pesoL\rangle\rangle, [(x, (y{*}2, 2046))|(x, y) \leftarrow \langle\langle PACIENTE, pesoQ\rangle\rangle])$

4.6.1.4 Conflitos de Precisão dos Dados

Neste exemplo, o atributo conceito têm seus valores representados como A, B, C ou D. O atributo nota tem seus valores representados como um número inteiro de 0 a 100. O conceito A equivale a notas maiores que 90, o conceito B equivale a notas maiores que 80 e menores que 90, o conceito C equivale a notas maiores que 70 e menores que 80 e

ALUNO (esquema de união)
cpf
nome
conceito

ALUNO (esquema local)
cpf
nome
nota

Tabela 4.5: Exemplo de Conflito de Precisão de Dados

o conceito D equivale a notas menores que 70. A resolução através do formalismo BAV seria feita assim:

- $addAtt(\langle\langle ALUNO, conceito\rangle\rangle, [(x, A)|(x, y) \leftarrow \langle\langle ALUNO, nota\rangle\rangle; y >= 90] + +[(x, B)|(x, y) \leftarrow \langle\langle ALUNO, nota\rangle\rangle; y >= 80; y < 90] + +[(x, C)|(x, y) \leftarrow \langle\langle ALUNO, nota\rangle\rangle; y >= 70; y < 80]) + +[(x, D)|(x, y) \leftarrow \langle\langle ALUNO, nota\rangle\rangle; y < 70]$

- $conAtt(\langle\langle ALUNO, nota\rangle\rangle, void)$

Mas este é um caso problemático em relação aos outros, pois sua resolução perde informações e consequentemente perde a propriedade de reversibilidade do formalismo.

4.6.1.5 Conflitos de Valores Padrão

PESSOAL (esquema de união)
cpf
nome
titulo

PESSOAL (esquema local)
cpf
nome
titulovp

Tabela 4.6: Exemplo de Conflito de Valores Padrão

Neste exemplo, o atributo titulo tem seus valores representados por ESTUDANTE, PROFESSOR ou FUNCIONÁRIO. No esquema local o atributo titulovp funciona da mesma forma, mas quando seu valor for ESTUDANTE, a ocorrência será representada com o valor VOID (NULL). A resolução através do formalismo BAV seria feita assim:

- $addAtt(\langle\langle PESSOAL, titulo\rangle\rangle, [(x, (if(y = void)(ESTUDANTE)(y)))|(x, y) \leftarrow \langle\langle PESSOAL, titulovp\rangle\rangle])$

- $delAtt(\langle\langle PESSOAL, titulovp\rangle\rangle, [(x, (if(y = ESTUDANTE)(void)(y)))|(x, y) \leftarrow \langle\langle PESSOAL, titulo\rangle\rangle])$

4.6.2 Conflitos entre Definições de Entidades

4.6.2.1 Conflitos de Equivalência de Chaves

ALUNO (esquema de união)
cpf
rg
nome

ALUNO (esquema local)
rg
cpf
nome

Tabela 4.7: Exemplo de Conflito de Equivalência de Chaves

Este é um caso simples, onde o identificador do esquema de união é um atributo no esquema local e vice-versa. Mas caso não seja possível encontrar uma função de mapeamento ou um identificador comum não será possível realizar a integração de modo satisfatório. A resolução através do formalismo BAV seria feita assim:

- $delPK(\langle\langle ALUNO, \langle\langle ALUNO, rg\rangle\rangle\rangle\rangle)$
- $addPK(\langle\langle ALUNO, \langle\langle ALUNO, cpf\rangle\rangle\rangle\rangle)$

4.6.2.2 Conflitos de União

ALUNO (esquema de união)
cpf
nome
rg

ALUNO (esquema local)
cpf
nome
altura

Tabela 4.8: Exemplo de Conflito de União

Neste exemplo, o atributo rg existe no esquema de união mas nunca existiu no esquema local. Já o atributo altura existe no esquema local mas não existe interesse em mantê-lo no esquema de união. A resolução através do formalismo BAV seria feita assim:

- $addAtt(\langle\langle ALUNO, rg\rangle\rangle, void)$
- $delAtt(\langle\langle ALUNO, altura\rangle\rangle, void)$

ALUNO (esquema de união)
cpf nomecompleto

ALUNO (esquema local)
cpf nome sobrenome

Tabela 4.9: Exemplo de Conflito de Isomorfismo

4.6.2.3 Conflitos de Isomorfismo

Neste exemplo, o esquema de união guarda o nome de um aluno em um atributo (nomecompleto) enquanto no esquema local o mesmo conceito é armazenado em dois atributos (nome e sobrenome). A função strConcat, responsável por concatenar duas strings, não existe na definição da linguagem IQL, mas as características da linguagem permitem que se defina novas funções. A resolução através do formalismo BAV seria feita assim:

- $addAtt(\langle\langle ALUNO, nomecompleto\rangle\rangle, [(x, strConcat(y, z))|(x, y) \leftarrow \langle\langle ALUNO, nome\rangle\rangle; (x, z) \leftarrow \langle\langle ALUNO, sobrenome\rangle\rangle])$
- $conAtt(\langle\langle ALUNO, nome\rangle\rangle, void)$
- $conAtt(\langle\langle ALUNO, sobrenome\rangle\rangle, void)$

Deve-se tomar cuidado uma vez que este é um caso problemático em relação aos outros. Dependendo da função de mapeamento escolhida pode ocorrer a perda da propriedade de reversibilidade do formalismo BAV.

4.6.2.4 Conflitos de Falta de Atributos

ALUNO (esquema de união)
cpf nome tipo

ALUNOGRAD (esquema local)
cpf nome

Tabela 4.10: Exemplo de Conflito de Falta de Dados

Neste exemplo, o esquema de união guarda alunos de todos os tipos (graduação, pós-graduação, etc.). Já o esquema local guarda apenas os alunos de graduação. Apesar

do atributo tipo não estar presente no esquema local, ele pode ser facilmente deduzido através de inferência. A resolução através do formalismo BAV seria feita assim:

- $addAtt(\langle\langle ALUNOGRAD, tipo\rangle\rangle, [(x, "grad")|x \leftarrow \langle\langle ALUNOGRAD\rangle\rangle])$
- $renRel(\langle\langle ALUNOGRAD\rangle\rangle, \langle\langle ALUNO\rangle\rangle)$

4.6.2.5 Conflitos entre Relações e Atributos

ALUNO (esquema de união)
cpf
nome
idade
orientador

ALUNO (esquema local)
cpf
nome
idade

ALUNOPG (esquema local)
cpf
orientador

Tabela 4.11: Exemplo de Conflito entre Relação e Atributo

Neste exemplo, o esquema de união guarda todas as informações sobre todos os alunos na mesma tabela. Já o esquema local guarda as informações gerais sobre os alunos em uma tabela e informações específicas sobre alunos da pós-graduação em outra tabela. A resolução através do formalismo BAV seria feita assim:

- $addAtt(\langle\langle ALUNO, orientador\rangle\rangle, [(x, y)|(x, y) \leftarrow \langle\langle ALUNOPG, orientador\rangle\rangle])$
- $delAtt(\langle\langle ALUNOPG, orientador\rangle\rangle, [(x, y)|(x, y) \leftarrow \langle\langle ALUNO, orientador\rangle\rangle])$
- $delRel(\langle\langle ALUNOPG\rangle\rangle, [(x)|(x, y) \leftarrow \langle\langle ALUNO, orientador\rangle\rangle; not(y = void)])$

4.6.2.6 Conflitos entre Atributos e Dados

ALUNO (esquema de união)
cpf
nome
tipo

ALUNO (esquema local)
cpf
nome
grad
mestre
doutor

Tabela 4.12: Exemplo de Conflito entre Atributo e Dados

Neste exemplo, o esquema de união guarda no atributo tipo a condição do aluno (grad, mestre, doutor). Já o esquema local possui um atributo do tipo boolean para cada condição possível. A resolução através do formalismo BAV seria feita assim:

- $addAtt(\langle\langle ALUNO, tipo\rangle\rangle, [(x, grad)|(x, TRUE) \leftarrow \langle\langle ALUNO, grad\rangle\rangle]{+}{+}[(x, mestre)|(x, TRUE) \leftarrow \langle\langle ALUNO, mestre\rangle\rangle]{+}{+}[(x, doutor)|(x, TRUE) \leftarrow \langle\langle ALUNO, doutor\rangle\rangle])$
- $delAtt(\langle\langle ALUNO, grad\rangle\rangle, [(x, TRUE)|(x, grad) \leftarrow \langle\langle ALUNO, tipo\rangle\rangle])$
- $delAtt(\langle\langle ALUNO, mestre\rangle\rangle, [(x, TRUE)|(x, mestre) \leftarrow \langle\langle ALUNO, tipo\rangle\rangle])$
- $delAtt(\langle\langle ALUNO, doutor\rangle\rangle, [(x, TRUE)|(x, doutor) \leftarrow \langle\langle ALUNO, tipo\rangle\rangle])$

CAPÍTULO 5

UMA PROPOSTA DE SISTEMA DE SOFTWARE PARA AUXÍLIO NA INTEGRAÇÃO DE BANCOS DE DADOS

O trabalho desenvolvido nessa dissertação alcança seu auge neste capítulo, onde é apresentada uma proposta de sistema de software para integração de bancos de dados através do formalismo BAV (seção 4.4). O sistema de software não foi implementado, uma vez que o objetivo era o de moldar sua concepção, seu funcionamento e suas necessidades, deixando a sua implementação para trabalhos futuros.

5.1 Visão Geral

Este sistema de software é utilizado na metodologia de integração que aplica o formalismo nas fontes de dados como se estas estivessem em uma linha de montagem (seção 4.5). Nessa metodologia assume-se um esquema de união e cada esquema local terá de se tornar compatível à união com o esquema assumido por meio de transformações BAV.

O objetivo deste sistema de software é auxiliar o usuário na geração das transformações BAV (seção 4.4) que tornarão a estrutura de um esquema local idêntica ao esquema de união. Para alcançar esse objetivo o sistema orienta o usuário através das várias etapas do processo consultando-o em momentos em que seja impossível para o sistema tomar uma decisão.

De modo sucinto, o que se espera do funcionamento do sistema de software proposto neste trabalho é que sejam fornecidos para ele dois esquemas, o primeiro relativo a um banco de dados local e o segundo relativo ao esquema de união definido para a integração. A medida que realiza diversas consultas ao usuário o software deve fornecer a sequência de transformações necessária para tornar a estrutura do esquema relativo ao banco de dados local idêntica a estrutura do esquema de união. A figura 5.1 ilustra esse processo mais nitidamente. Baseado nessas características de funcionamento o sistema de software

foi batizado como: *Sistema de Software para Auxílio na Geração de Transformações BAV para Integração de Bancos de Dados*, abreviado através da sigla *SAI*.

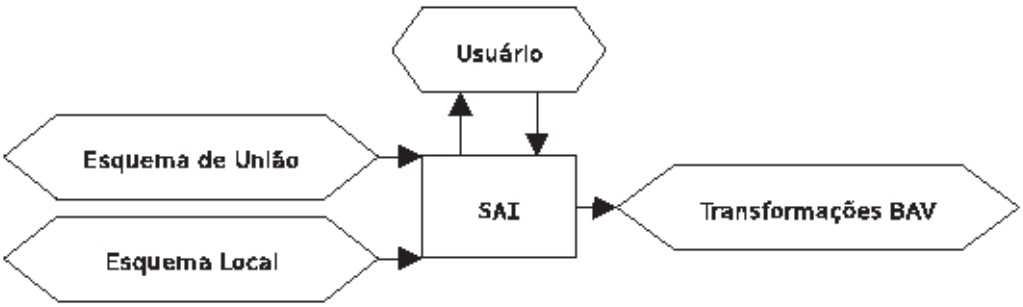

Figura 5.1: Funcionamento do Sistema SAI

Para gerar os esquemas de entrada em um formato aceitável foi concebida superficialmente a idéia de um *Software para Geração dos Esquemas de Entrada*, ou simplesmente *SGEE*. O objetivo deste sistema de software é prover o usuário com ferramentas que permitam a descrição dos objetos de um esquema de banco de dados (relações, atributos, chaves primárias e chaves estrangeiras) e gere um arquivo onde seus objetos sejam representados através de algum tipo de estrutura de dados compreendida pelo SAI. A representação escolhida para os objetos do esquema ainda não foi definida. Essa escolha será postergada para uma futura implementação. Entre as possíveis estruturas imaginadas estão grafos, matrizes ou XML.

Tanto o esquema de união quanto o esquema local devem ser gerados através do SGEE. A definição do esquema de união deverá ser feita pelo líder do projeto de integração. A definição do esquema do banco de dados local já existe e deverá ser traduzida através do SGEE pelo seu administrador.

O usuário do SAI deve estar familiarizado com o esquema de união, com o esquema local que vai ser integrado e com o formalismo BAV. A primeira sugestão para o usuário ideal do SAI seria o administrador do banco de dados local, uma vez que normalmente é ele quem possui mais conhecimentos a respeito do esquema local. Conhecimentos sobre o esquema de união podem ser repassados para ele através do líder do projeto de integração. O único problema são os conhecimentos a respeito do formalismo BAV. Caso não seja do interesse do administrador do banco de dados local aprender o formalismo BAV ele deverá

trabalhar em equipe com alguém que o conheça.

A figura 5.2 exibe um esboço de como seria a interface visual do programa. Pode-se perceber quatro áreas distintas na figura. Elas são nomeadas como área do esquema de união, área do esquema local, área das transformações e área de diálogo.

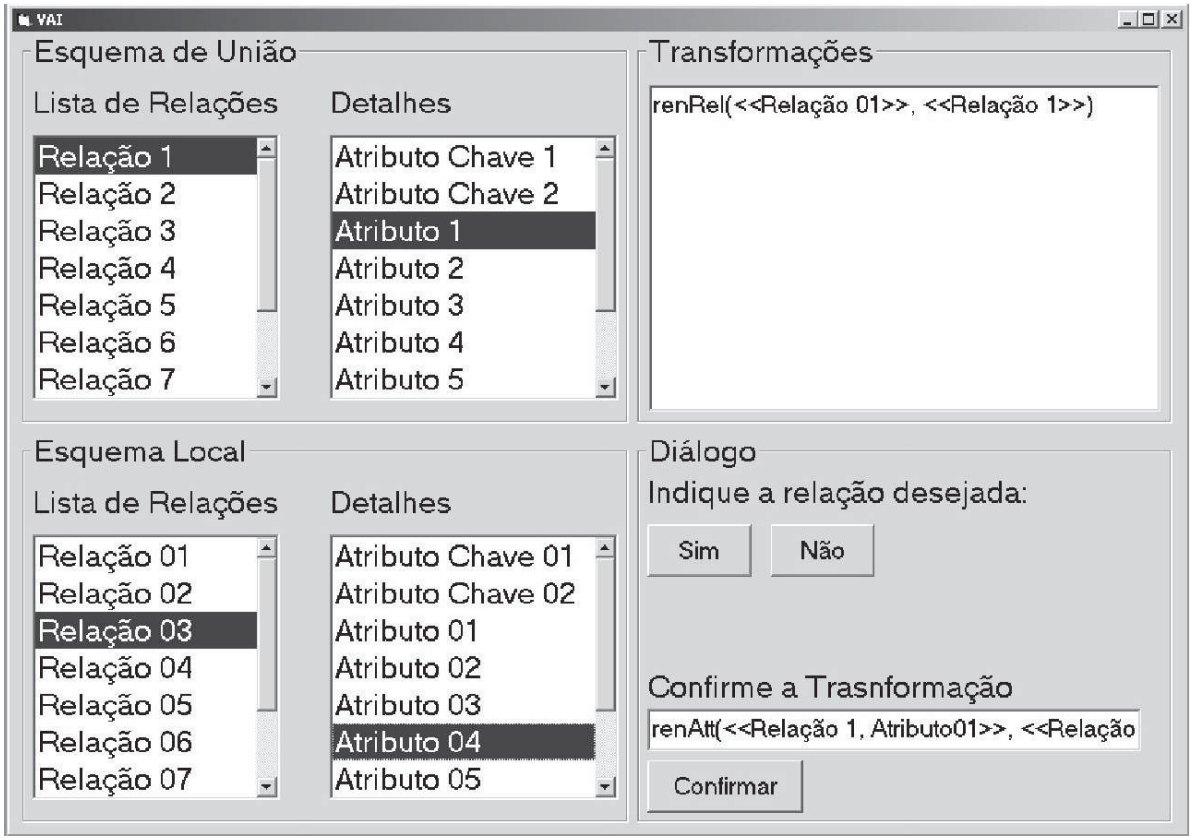

Figura 5.2: Visualização do Sistema SAI

Quando o SAI for inicializado com seus esquemas de entrada eles serão exibidos respectivamente nas áreas do esquema de união e do esquema local. O usuário poderá navegar tranquilamente através dos esquemas quando isso for necessário para resolver qualquer tipo de dúvida quanto a estrutura dos esquemas. As consultas ao usuário serão feitas através da área de diálogo. Essas consultas poderão pedir para que ele indique alguma relação ou atributo específico no esquema local, o que será feito com um clique na relação ou atributo na área do esquema local, ou poderão pedir que o usuário revise e confirme alguma transformação BAV que esteja sendo gerada. Após ser confirmada cada transformação é armazenada na área de transformações e seu efeito é aplicado na área do esquema local de modo que o usuário possa visualizar o que a transformação causou.

Ao término da execução do SAI os esquema de entrada serão idênticos e o conteúdo da área de transformações será o resultado do processo de integração.

5.2 O Funcionamento do SAI

Antes de explicar o funcionamento deve-se esclarecer alguns conceitos definidos para este sistema de software.

Dois objetos são *equivalentes* quando guardam o mesmo conceito semântico com o mesmo formato de dados de modo que possam ser submetidos a união sem qualquer tipo de problema.

A cada momento que o sistema descobrir que dois atributos são equivalentes, sejam eles atributos chaves ou não, eles serão renomeados para que tenham o mesmo nome e então *unificados*, ou seja, anotados em uma *tabela de unificações* para que o sistema saiba que não deve mais se ocupar com eles. Cada anotação na tabela de unificações será composta de um par, sendo o primeiro elemento um atributo de uma relação do esquema de união e o segundo elemento um atributo de uma relação do esquema local.

A partir desses conceitos as atividades do SAI foram divididas em etapas distintas conforme suas características. São elas:

1. Remover Chaves Estrangeiras
2. Gerar Tabela de Correlacionamentos
3. Unificar Chaves Primárias
4. Unificar Atributos
5. Remover Objetos não Unificados
6. Gerar Chaves Estrangeiras

Cada uma dessas etapas será explicada a seguir. Para aqueles que queiram se aprofundar mais no funcionamento de cada uma delas foi construída uma especificação informal para cada uma dessas etapas. Estas especificações podem ser encontradas no Anexo B.

5.2.1 Remover Chaves Estrangeiras

Uma vez que o esquema local se tornará compatível à união com o esquema de união é intuitivo que só se manterão no esquema local chaves estrangeiras que estejam ligadas a relações cujos conceitos semânticos também estejam no esquema de união.

Mas chaves estrangeiras são apenas restrições que não carregam nenhum tipo de instância. Então, ao invés do sistema se dedicar a distinguir quais chaves estrangeiras devem ser removidas do esquema local e quais não, o sistema remove todas as chaves estrangeiras e as adiciona novamente conforme o esquema de união quando o processo estiver terminado.

A maior vantagem dessa atitude é a perda de complexidade do sistema ao lidar com chaves estrangeiras. A maior desvantagem dessa atitude é a possível geração de transformações redundantes, uma vez que pode ocorrer de uma chave estrangeira ser removida e gerada novamente da mesma forma. Mesmo assim a vantagem supera a desvantagem tornando esse um preço pequeno a se pagar.

5.2.2 Gerar Tabela de Correlacionamentos

Esta etapa têm por objetivo decidir através de consultas ao usuário o que será feito com cada uma das relações do esquema local durante o restante do processo. Existem apenas duas opções para cada relação: ou é aplicada uma sequência de transformações à relação de modo que ela se torne equivalente a uma relação do esquema de união ou então as instâncias relevantes da relação do esquema local serão transferidas para outras relações antes que ela seja apagada.

Tendo isso em vista foi criado o conceito da *tabela de correlacionamentos*. Essa tabela representa uma função parcial e injetora tendo como domínio as relações do esquema de união e tendo como imagem as relações do esquema local.

Para preencher a tabela de correlacionamentos é feita a seguinte pergunta ao usuário para cada relação do esquema de união: caso exista, indique a relação do esquema local que abriga os conceitos semânticos mais próximos daqueles guardados pela relação do esquema de união. Desse modo é criado um mapeamento onde cada elemento do conjunto

de relações do esquema de união aponta para um elemento distinto do conjunto de relações do esquema local ou então não aponta para ninguém.

Nas etapas seguintes será criada uma nova relação no esquema local para cada relação do esquema de união cuja entrada na tabela de correlacionamentos não aponte para ninguém. Cada relação do esquema local que participe da tabela de correlacionamentos terá uma sequência de transformações aplicada a ela. Cada relação do esquema local que não participe da tabela de correlacionamentos será apagada depois que as instâncias relevantes tenham sido transferidas para outras relações.

Para auxiliar o usuário durante a sua tarefa de responder as perguntas, o sistema emprega a *função Adivinho*. Para cada relação do esquema de união a função devolve uma relação do esquema local que julgue ser a melhor resposta para a pergunta feita ao usuário. Essa função seria utilizada para dar uma sugestão de resposta ao usuário.

Infelizmente não foi possível definir com precisão essa função devido a falta de acesso a esquemas de bancos de dados construídos em ocasiões distintas para um propósito similar. Esses bancos de dados serviriam para testar e validar os parâmetros de decisão da função adivinho. Mas entre os parâmetros sugeridos para essa função estariam:

- similaridade entre os nomes das relações;
- similaridade entre os nomes dos atributos chaves;
- número de atributos chaves;
- tipos dos atributos chaves;
- número de atributos.

5.2.3 Unificar Chaves Primárias

Esta etapa tem por objetivo gerar transformações que tornem o conjunto de chaves primárias das relações do esquema local equivalentes as chaves primárias das relações do esquema de união baseando-se na tabela de correlacionamentos.

Cada relação do esquema de união será submetida a esta etapa seguindo uma ordem estipulada pela *função PróximaRelação*. Essa função toma o esquema de união e a tabela de unificações como parâmetros de entrada e devolve uma de suas relações como parâmetro de saída. Ela utiliza as chaves estrangeiras definidas no esquema de união para fazer sua escolha. Basicamente ela irá devolver uma relação do esquema de união que seja filha do menor número de relações que ainda não tenham passado por essa etapa. Esta política mostrou-se bastante útil para evitar a geração de transformações desnecessárias e simplicação das mesmas.

Esta etapa é composta de duas etapas menores: *resolução de conflitos entre chaves primárias* e *extração de chaves primárias*. Essas etapas menores serão utilizadas conforme o estado de cada relação do esquema de união em relação a tabela de correlacionamentos.

Se uma relação do esquema de união possuir uma relação do esquema local correspondente na tabela de correlacionamentos será ativada a etapa *resolução de conflitos entre chaves primárias* que irá gerar transformações que alterem a relação do esquema local.

O primeiro passo é gerar uma transformação que renomeie a relação do esquema local de modo que ela fique com o mesmo nome da relação do esquema de união. A seguir o usuário deverá ser consultado sobre a equivalência do conjunto de atributos chave das relações. Se forem equivalentes basta renomeá-los conforme a necessidade e unificá-los. Caso contrário serão feitas perguntas ao usuário sobre a origem das instâncias de cada um dos atributos chave com o objetivo de gerar transformações que os adicionem a relação do esquema local. Essas transformações serão apresentadas ao usuário porque resolvem as situações mais simples mas precisam ser revisadas para que se adaptem aos casos de conflito mais complicados. Por fim, se necessário, serão geradas transformações que apaguem e recriem a restrição de chave primária.

Mas se uma relação do esquema de união não possuir uma relação do esquema local correspondente na tabela de correlacionamentos será ativada a etapa *extração de chaves primárias* que irá gerar transformações que criem uma nova relação no esquema local. Por enquanto esta relação terá apenas seus atributos chave e esses serão equivalentes aos atributos chave do esquema de união. Essa nova relação ocupará o espaço vazio na tabela

de correlacionamentos.

O usuário deverá indicar, se existir, a origem das instâncias do conjunto de atributos chave com o objetivo de gerar transformações que resolvem as situações mais simples mas que precisam ser revisadas pelo usuário para se adaptarem aos casos de conflito mais complicados. Caso a origem dessas instâncias não exista serão geradas transformações que criem a relação com um conjunto vazio de instâncias. Por fim serão geradas transformações que criem a restrição de chave primária.

5.2.4 Unificar Atributos

Nas etapas anteriores foram geradas transformações que adicionaram relações ao esquema local de modo que a função parcial representada pela tabela de correlacionamentos se tornasse uma função total. Agora cada relação do esquema de união possui uma relação correspondente no esquema local e todas as chaves primárias foram unificadas.

O objetivo desta etapa é gerar transformações que determinem, para cada atributo que não seja uma chave primária em cada relação do esquema de união, um atributo no esquema local que seja equivalente ao atributo do esquema de união.

A escolha da ordem em que as relações do esquema de união são submetidas é definida pela *função PróximaRelação* adaptada para esta etapa, mas com o mesmo funcionamento. Uma vez escolhida a relação do esquema de união a ser trabalhada o sistema consulta a tabela de correlacionamentos para saber qual é a relação do esquema local correspondente.

Então, para cada atributo da relação do esquema de união, o usuário deverá ser consultado sobre a existência de um atributo na relação do esquema local que seja equivalente ao atributo do esquema de união. Se esse atributo existir basta renomeá-lo conforme a necessidade e unificá-lo. Caso contrário o usuário será consultado sobre a origem das instâncias desse atributo com o objetivo de gerar transformações que o adicionem a relação do esquema local. Essas transformações serão apresentadas ao usuário porque resolvem as situações mais simples mas precisam ser revisadas para que se adaptem aos casos de conflito mais complicados. Caso a origem das instâncias de um atributo não exista no esquema local o sistema gerará uma transformação que gere o atributo sem instâncias.

Ao fim do processo para cada atributo ele deve ser unificado com seu correspondente no esquema local.

5.2.5 Remover Objetos não Unificados

Atualmente cada atributo de cada relação do esquema de união, seja ele uma chave primária ou não, está unificado com um atributo do esquema local. O objetivo desta etapa é gerar transformações que retirem do esquema local os atributos que não foram unificados e as relações que não possuam atributos unificados.

Tudo que o sistema precisa fazer é gerar a transformação e entregá-la ao usuário para que ele a revise escrevendo a consulta que mostra como recuperar as instâncias do objeto que está sendo removido. Mas essa parte do processo pode ser melhorada para o usuário através de sugestões de consultas. Para obter essas sugestões surgiu o conceito de um *reversor de consultas*.

O reversor de consultas pode ser explicado, de modo sucinto, através de um pequeno exemplo. Suponha uma transformação genérica que adiciona um atributo, por exemplo. Ela se apresenta na forma:

$$addAtt(\langle\langle\ objeto\ destino\ \rangle\rangle, [(\ tupla\ destino\ )\ |\ (\ tupla\ origem\ ) \leftarrow \langle\langle\ objeto\ origem\ \rangle\rangle; (\ outras\ tuplas\ ) \leftarrow \langle\langle\ outros\ objetos\ \rangle\rangle;\ restrições\ lógicas\ ])$$

O objeto destino é o atributo que está sendo adicionado a uma relação do esquema. A tupla destino representa tal atributo juntamente com quaisquer funções a serem aplicadas em seus valores. O objeto origem representa o atributo do qual as instâncias estão sendo retiradas para preencherem o atributo destino. A tupla origem está intrinsicamente ligada a tupla destino uma vez que são seus valores que fazem a conexão entre o objeto destino e o objeto origem. Outras tuplas, outros objetos e restrições lógicas são utilizados para filtrar as instâncias na transformação.

É muito frequente surgir a necessidade de se retirar do esquema o objeto origem para alguma transformação que já tenha sido gerada anteriormente. Após alguns estudos pôde-se verificar que na maior parte dos casos simples tudo o que se tem de fazer é gerar uma

transformação da forma:

$$delAtt(\langle\langle\ objeto\ origem\ \rangle\rangle, [(\ tupla\ origem\ )\ |\ (\ tupla\ destino\ ) \leftarrow \langle\langle\ objeto\ destino\ \rangle\rangle])$$

O tipo da transformação foi invertido. Os objetos e tuplas de origem e de destino foram invertidos. As funções que eram aplicadas à tupla destino agora têm sua função inversa aplicadas a tupla origem. As outras tuplas, os outros objetos e as restrições lógicas foram ignoradas porque sua aplicação só tinha valor na transformação anterior.

A consulta resultante foi batizada de "consulta reversa"da consulta original e o mecanismo que a constrói foi batizado de reversor de consultas.

Desse modo antes de entregar uma transformação que retire algum objeto do esquema para o usuário o sistema pode buscar nas transformações geradas nas fases anteriores uma transformação que permita gerar a transformação reversa que se adeque as necessidades do usuário.

Infelizmente não foi possível construir o reversor de consultas nem especificá-lo com precisão nessa dissertação devido a extensão do trabalho já apresentado e o tempo necessário para desenvolvê-lo, restando apenas deixá-lo para trabalhos futuros.

5.2.6 Gerar Chaves Estrangeiras

Nesta etapa o processo de geração de transformações está praticamente completo. Todos os atributos do esquema de união já foram unificados com os atributos do esquema local e todos os atributos que não haviam sido unificados com ninguém já foram retirados do esquema.

Tudo o que resta a fazer é gerar as transformações que recriem as chaves estrangeiras no esquema local de modo que elas fiquem idênticas ao esquema de união.

Por fim, a sequência de transformações geradas durante todas as etapas anteriores são o resultado do processo sendo suficientes para tornar a estrutura do esquema local idêntica a estrutura do esquema de união.

5.3 Vantagens do Sistema SAI

O SAI foi concebido com o objetivo de auxiliar um administrador a escrever as trasnformações BAV necessárias para tornar dois esquemas de bancos de dados compatíveis a união. Infelizmente resultados como a automatização completa do processo sem a participação humana ou mesmo uma participação humana que não necessitasse de conhecimentos profundos sobre os esquemas ou sobre o formalismo BAV não puderam ser alcançados.

Mas essa proposta de sistema de software SAI traz muitas vantagens ao usuário, como por exemplo:

- orientação do processo de geração de transformações, dividindo-o por etapas e organizando-o de modo a torná-lo mais simples e fácil de entender;
- agilidade no processo de escrita, o usuário tem de pensar apenas nos casos que o sistema não pode decidir sozinho;
- geração automatica de transformações para os casos mais simples;
- geração de sugestões para os casos mais complexos, fornecendo a solução parcial ao usuário para revisão;
- interface visual e amigavel para o usuário.

CAPÍTULO 6

CONCLUSÃO

O dos maiores objetivos da integração de bancos de dados é fornecer uma interface integrada e uniforme para consultas em diversos bancos de dados que descrevam os mesmos objetos do mundo real. Assim um sistema de integração de dados possibilita ao usuário se concentrar no que ele deseja sem ter de pensar em como conseguir essa informação.

De modo a permitir que o sistema de integração de banco de dados possa responder as consultas, deve haver alguma descrição do relacionamento entre o esquema global e os esquemas locais. O processador de consultas deve estar apto a reformular uma consulta submetida a ele como novas consultas submetidas aos esquemas locais [9].

O objetivo desta dissertação foi o de propor um sistema de software que automatize parcialmente o processo de integração de bancos de dados através da geração de transformações do formalismo BAV.

Mas antes de chegar a esse objetivo foram geradas contribuições menores como:

- foram estudadas diferentes visões do processo de integração de bancos de dados unindo várias de suas fases de modo a se obter uma divisão mais completa de suas etapas;

- foi realizado um estudo sobre os casos de conflito entre esquemas existentes, apresentando os resultados através de exemplos no modelo relacional de modo a tornar seu entendimento mais claro;

- para cada um dos casos de conflito existentes foram apresentadas possíveis soluções através do formalismo BAV com o objetivo de validar sua capacidade de resolver tais conflitos;

- uma vez que nenhuma das metodologias utilizadas pelo formalismo BAV é bem descrita no material já publicado, o autor desta dissertação produziu um texto que

explica com maiores detalhes essas metodologias de modo a tornar claro o papel do sistema de software no processo de integração de bancos de dados.

O objetivo do sistema de software proposto foi auxiliar o usuário na geração das transformações BAV que tornarão a estrutura de um esquema local idêntica ao esquema de união. Para alcançar esse objetivo o sistema orienta o usuário através das várias etapas do processo consultando-o em momentos em que seja impossível para o sistema tomar uma decisão.

Este sistema de software foi batizado como *Sistema de Software para Auxílio na Geração de Transformações BAV para Integração de Bancos de Dados*, abreviado através da sigla *SAI*.

Até onde se tem conhecimento, o sistema de softwarte SAI é a primeira tentativa de automatização parcial da geração de transformações BAV no processo de integração de bancos de dados.

Este sistema traz muitas vantagens ao usuário encarregado da tarefa de integração, tais como:

- orientação do processo de geração de transformações, dividindo-o por etapas e organizando-o de modo a torná-lo mais simples e fácil de entender;
- agilidade no processo de escrita, o usuário tem de pensar apenas nos casos em que o sistema não pode decidir sozinho;
- geração automatica de transformações para os casos mais simples;
- geração de sugestões para os casos mais complexos, fornecendo a solução parcial ao usuário para revisão;
- interface visual amigavel para o usuário.

Mas esta proposta deixou questões em aberto, que deverão ser realizadas em trabalhos futuros. São elas:

- finalização da especificação do *Software para Geração dos Esquemas de Entrada*, ou simplesmente *SGEE*, bem como a sua implementação.

- testar os parâmetros da função adivinho de modo a otimizar o seu resultado, melhorando assim as sugestões dadas pelo SAI;
- finalização da especificação *reversor de consultas* bem como a sua implementação;
- implementar o sistema de software SAI, realizar testes e apresentar seus resultados.

BIBLIOGRAFIA

[1] Serge Abiteboul, Richard Hull, e Victor Vianu. *Foundations of Databases*. Addison-Wesley Publishing Company, 1995.

[2] C. Batini, M. Lenzerini, e S. B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys (CSUR)*, 18(4):323–364, 1986.

[3] M. Boyd, P. McBrien, e N. Tong. The automed schema integration repository. *19th British National Conference on Databases (BNCOD19), Lecture Notes in Computer Science (LNCS)*, 2405:42–45, 2002.

[4] Michael Boyd, Charalambos Lazanitis, Sasivimol Kittivoravitkul, Peter McBrien, e Nikos Rizopoulos. An overview of the automed repository. Relatório técnico, AutoMed Project, 2004.

[5] Peter Buneman, Leonid Libkin, Dan Suciu, Val Tannen, e Limsoon Wong. Comprehension syntax. *SIGMOD Record*, 23(1):87–96, 1994.

[6] Sudarshan Chawathe, Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey D. Ullman, e Jennifer Widom. The TSIMMIS project: Integration of heterogeneous information sources. *16th Meeting of the Information Processing Society of Japan*, páginas 7–18, Tokyo, Japan, 1994.

[7] E. Jasper, A. Poulovassilis, e L. Zamboulis. Processing iql queries and migrating data in the automed toolkit. Relatório técnico, AutoMed Project, 2003.

[8] Won Kim e Jungyun Seo. Classifying schematic and data heterogeneity in multidatabase systems. *Computer*, 24(12):12–18, 1991.

[9] Alon Y. Levy. Logic-based techniques in data integration. Jack Minker, editor, *Workshop on Logic-Based Artificial Intelligence, Washington, DC, June 14–16, 1999*, College Park, Maryland, 1999. Computer Science Department, University of Maryland.

[10] Alon Y. Levy, Anand Rajaraman, e Joann J. Ordille. Querying heterogeneous information sources using source descriptions. *Proceedings of the Twenty-second International Conference on Very Large Databases*, páginas 251–262, Bombay, India, 1996. VLDB Endowment, Saratoga, Calif.

[11] P. McBrien. The university database integration: An automed example. Relatório técnico, AutoMed Project, 2003.

[12] P. McBrien e A. Poulovassilis. A general formal framework for schema transformation. Relatório Técnico 98-05, King's College London, 1998.

[13] P. McBrien e A. Poulovassilis. Data integration by bi-directional schema transformation rules. 19th International Conference on Data Engineering, 2003.

[14] P. McBrien, A. Poulovassilis, N. Tong, e E. Jasper. View generation and optimisation in the automed data integration framework. Relatório técnico, AutoMed Project, 2003.

[15] Peter McBrien e Alexandra Poulovassilis. A formalisation of semantic schema integration. *Information Systems*, 23(5):307–334, 1998.

[16] Peter McBrien e Alexandra Poulovassilis. Automatic migration and wrapping of database applications - a schema transformation approach. *International Conference on Conceptual Modeling / the Entity Relationship Approach*, páginas 96–113, 1999.

[17] Peter McBrien e Alexandra Poulovassilis. A uniform approach to inter-model transformations. *Conference on Advanced Information Systems Engineering*, páginas 333–348, 1999.

[18] Christine Parent e Stefano Spaccapietra. Issues and approaches of database integration. *Commun. ACM*, 41(5es):166–178, 1998.

[19] A. Poulovassilis. The automed intermediate query language. Relatório técnico, AutoMed Project, 2001.

[20] A. Poulovassilis. A tutorial on the iql query language. Relatório técnico, AutoMed Project, 2004.

[21] Amit P. Sheth e Vipul Kashyap. So far (schematically) yet so near (semantically). *Proceedings of the IFIP WG 2.6 Database Semantics Conference on Interoperable Database Systems (DS-5)*, páginas 283–312. North-Holland, 1993.

[22] Abraham Silberschatz, Henry F. Korth, e S. Sudarshan. *Database System Concepts*. WCB/McGraw-Hill, 1998.

[23] Stefano Spaccapietra, Christine Parent, e Yann Dupont. Model independent assertions for integration of heterogeneous schemas. *The VLDB Journal*, 1(1):81–126, 1992.

[24] Jeffrey D. Ullman. *Principles of Database Systems*. Computer Science Press, 1982.

[25] Jeffrey D. Ullman. *Principles of Database and Knowledge - Base Systems*. Computer Science Press, 1988.

[26] Gottfried Vossen. *Data Models, Database Languages and Database Management Systems*. Addison-Wesley Publishing Company, 1990.

ANEXO A

A.1 O Modelo de Dados Baseado em Hipergrafos

A.1.1 Noções Básicas

Muitos formalismos e metodologias já foram criados para resolver o problema da integração de bancos de dados. A. Poulovassilis e P. McBrien criaram um formalismo [12] que procura eliminar as diferenças entre esquemas através de transformação dos mesmos. Este formalismo é denominado *modelo de dados baseado em hipergrafos (hipergraph data model - HDM).*

Suponha a existência de dois conjuntos disjuntos: o conjunto $Value$, composto por todos os valores, e o conjunto $Name$, composto por todos os nomes que um vértice ou uma aresta podem receber.

O conjunto $Scheme$ é definido recursivamente da seguinte maneira:

- $Name \subseteq Scheme$;
- $\langle n_0, n_1, \ldots, n_m \rangle \in Scheme$ *se* $m \geq 1, n_0 \in Name$ *e* $n_i \in Scheme$ *para todo* $1 \leq i \leq m$.

A constante $Null \in Name$ (também chamada de $Void$). Para qualquer conjunto T, $Seq(T)$ indica o conjunto de sequências finitas construída com membros de T.

Desse modo, um *Esquema* S é uma tripla $\langle Nodes, Edges, Constraints \rangle$ (representando respectivamente o conjunto dos vértices, arestas e restrições) onde:

- $Nodes \subseteq Name$;
- $Edges \subseteq Name\ X\ Seq(Scheme)$ de modo que para todo $\langle n_0, n_1, \ldots, n_m \rangle \in Edges, n_i \in Nodes \cup Edges$ para todo $1 \leq i \leq m$;
- $Constraints$ é um conjunto de expressões que resultam valores booleanos e cujas variáveis pertencem ao conjunto $Nodes \cup Edges$.

Desse modo, $Nodes$ e $Edges$ definem um hipergrafo aninhado e rotulado. Aninhado porque uma aresta pode se ligar a qualquer número de vértices e/ou outras arestas. Vértices são identificados unicamente por seus nomes. Arestas e restrições podem ter um nome opcional associado a elas. A linguagem utilizada por restrições ou por quaisquer consultas direcionadas ao esquema não está associada ao formalismo de modo a dar-lhe mais liberdade.

Dado um esquema $S = \langle Nodes, Edges, Constraints \rangle$, uma instância de S é um conjunto I tal que $I \subseteq P(Seq(Value))$ e exista uma função $Ext_{S,I} : Nodes \cup Edges \longrightarrow P(Seq(Value))$ onde:

- cada conjunto pertencente a imagem de $Ext_{S,I}$ pode ser derivado através de uma expressão sobre os conjuntos de I;
- cada conjunto em I pode ser derivado através de uma expressão sobre os conjuntos da imagem de $Ext_{S,I}$;
- para cada aresta $\langle n_0, c_1, \ldots, c_m \rangle \in Edges$, cada sequência $s \in Ext_{S,I}(\langle n_0, c_1, \ldots, c_m \rangle)$ contém m subsequências $s_1, \ldots, sm$ onde $s_i \in Ext_{S,I}(c_i)$ para todo $1 \leq i \leq m$ (integridade de domínio);
- para todo $c \in Constraints$, a expressão $c[c_1/Ext_{S,I}(c_1), \ldots, c_n/Ext_{S,I}(c_n)]$ retorna verdadeiro, onde $c_1, \ldots, c_n$ são elementos de $Nodes \cup Edges$.

Tal função $Ext_{S,I}$ é chamada de *mapeamento de extensões* de S para I.

Um *modelo* é uma tripla $\langle S, I, Ext_{S,I} \rangle$ onde S é um esquema, I é uma instância de S e $Ext_{S,I}$ é uma função de mapeamento de extensões de S para I. O conjunto dos modelos é chamado de $Models$.

Dois esquemas são *equivalentes* se eles têm o mesmo conjunto de instâncias. Dada uma condição f, um esquema S condicionalmente contém um esquema S' com respeito a f se qualquer instância de S' satisfazendo f também é uma instância de S. Dois esquemas S e S' são *condicionalmente equivalentes* com respeito a f se ambos condicionalmente contém um ao outro com respeito a f.

A.1.2 Transformações Primitivas

As transformações primitivas suportadas pelo HDM são listadas a seguir. Cada uma das transformações é uma função que quando aplicada a um modelo retorna um novo modelo. Cada transformação possui uma exigência que deve se manter para a tranformação resultar em sucesso. Transformações que falham retornam um modelo indefinido (ou inválido) representado por ϕ. Qualquer transformação aplicada a ϕ retorna ϕ.

- $renameNode(fromName, toName)$ - Renomeia um vértice. Exigência: não deve existir um vértice já com nome igual a $toName$.
- $renameEdge(\langle fromName, c_1, \ldots, c_m \rangle, toName)$ - Renomeia uma aresta. Exigência: não deve existir uma aresta já com nome igual a $toName$.
- $addConstraint\ c$ - Adiciona uma nova restrição c. Exigência: c deve retornar verdadeiro.
- $delConstraint\ c$ - Apaga uma restrição c. Exigência: c deve existir.
- $addNode(name, q)$ - Adiciona um vértice chamado $name$ cuja extensão é dada pela consulta q. Exigência: não deve existir um vértice já com nome igual a $name$.
- $delNode(name, q)$ - Apaga um vértice chamado $name$. q é um consulta que determina como a extensão do vértice apagado pode ser recuperada a partir dos objetos restantes do esquema. Exigência: o vértice deve existir e não deve participar de nenhuma aresta.
- $addEdge(\langle name, c_1, \ldots, c_m \rangle, q)$ - Adiciona uma nova aresta entre uma sequência de objetos $c_1, \ldots, c_m$ do esquema. A extensão da aresta é determinada pelo valor da consulta q. Exigência: a aresta já não existe, $c_1, \ldots, c_m$ existem e q satisfaz as restrições de domínio apropriadas.
- $delEdge(\langle name, c_1, \ldots, c_m \rangle, q)$ - Apaga uma aresta. q é um consulta que determina como a extensão da aresta apagada pode ser recuperada a partir dos objetos restantes do esquema. Exigência: a aresta existe e não participa de outras arestas.

Para cada uma dessas transformações existe uma outra versão que toma um argumento extra. Este argumento é uma condição que deve ser satisfeita para que a transformação tenha sucesso.

Uma *transformação composta* é uma sequência de $n \geq 1$ transformações primitivas. Uma transformação t é *dependente do esquema (schema-dependent - s-d)* com respeito ao esquema S se não retornar ϕ para qualquer modelo de S, caso contrário t é *dependente da instância (instance-dependent - i-d)* com respeito a S. É facil perceber que se um esquema S pode ser transformado em um esquema S' através de transformações dependentes do esquema e vice-versa, então S e S' são equivalentes. Se S pode ser transformado em S' através de transformações dependentes da instância com uma exigência f e vice-versa, então S e S' são condicionalmente equivalentes com respeito a f.

Para cada transformação primitiva t tal que $t((S, I, Ext_{S,I})) \neq \phi$ existe uma *transformação primitiva inversa* t^{-1} tal que $t^{-1}(t((S, I, Ext_{S,I}))) = (S, I, Ext_{S,I})$. No caso de t possuir um argumento extra c para validar a transformação ele não precisa estar em t^{-1} uma vez que ele foi validado em t. Uma listagem das transformações inversas é apresentada na tabela 6.1.

Transformação t	Transformação t^{-1}
$renameNode(from, to)\ c$	$renameNode(to, from)$
$renameEdge(\langle from, schemes\rangle, to)\ c$	$renameEdge(\langle to, schemes\rangle, from)$
$addConstraint\ q\ c$	$delConstraint\ q$
$delConstraint\ q\ c$	$addConstraint\ q$
$addNode(n, q)\ c$	$delNode(n, q)$
$delNode(n, q)\ c$	$addNode(n, q)$
$addEdge(e, q)\ c$	$delEdge(e, q)$
$delEdge(e, q)\ c$	$addEdge(e, q)$

Tabela 6.1: Transformações Inversas do Formalismo HDM

A reversibilidade das transformações primitivas pode ser generalizada para qualquer transformação composta que tenha resulte em sucesso: dada uma tranformação composta $t_1; \ldots; t_n$ a sua transformação inversa é $t_n^{-1}; \ldots; t_1^{-1}$.

Mesmo quando dois bancos de dados são designados para guardar a mesma informação eles provavelmente vão ser um pouco diferentes e não serão exatamente equivalentes. Para

lidar com tais casos foram criadas [16] mais quatro transformações para lidar com tais situações. Elas são definidas a partir das transformações primitivas existentes na tabela 6.2.

$extendNode\ n = addNode(n, VOID)$
$extendEdge\ n = addEdge(e, VOID)$
$contractNode\ n = delNode(n, VOID)$
$contractEdge\ n = delEdge(e, VOID)$

Tabela 6.2: Transformações Adicionais do Formalismo HDM

Mais informações sobre a utilização e evolução deste formalismo podem ser encontradas em [15] onde foi demonstrado que o formalismo consegue suportar todas as trasnformações usadas na integração de esquemas utilizando o modelo de dados entidade-relacionamento e em [17] onde foi demonstrado que o formalismo pode ser aplicado para várias linguagens de modelagem de alto nível como o modelo de dados entidade-relacionamento, relacional, documentos da Internet e até mesmo UML.

ANEXO B

B.1 Observações Gerais

A seguir serão apresentadas especificações informais de cada uma das etapas do sistema de software SAI. Maiores explicações sobre cada uma dessas etapas podem ser encontradas no capítulo 5.

Letras do alfabeto escritas em letra maiúsculas representam relações. Quando acompanhadas do sinal ' representam um atributo da relação. Quando utilizadas dentro de transformações representam o nome do objeto e não o objeto em si. Quando utilizadas fora das transformações representam o objeto em si e não apenas seu nome.

Foram definidas algumas abreviaturas nas especificações para as palavras que ocorriam com mais frequência. São elas:

- rel. = relação
- rels. = relações
- esq. = esquema
- atr. = atributo
- atrs. = atributos

B.2 Etapas do SAI

B.2.1 Remover Chaves Estrangeiras

Para cada chave estrangeira com uma rel. filha A e uma rel. pai B no esq. local:

Gerar transformação sem revisão do usuário:
delFK(<< rel. A, <<rel. A, atrs. da chave estrangeira na rel. A>>, rel. B, <<rel. B, atrs. da chave estrangeira na rel. B>>>>)

B.2.2 Gerar Tabela de Correlacionamentos

Para cada rel. A no esq. de união:

Aplicar a função adivinho fornecendo a rel. A e obtendo uma rel. B do esq. local.

Perguntar ao usuário qual é a rel. do esq. local que ele escolhe para fazer o correlacionamento, fornecendo como sugestão a rel. B.

Adicionar a informação na tabela de correlacionamentos.

B.2.3 Unificar Chaves Primárias

Enquanto existirem rels. no esq. de união com chaves primárias não unificadas:

Aplicar a função PróximaRelação para escolher uma rel. A do esq. de união.

Verificar se existe uma rel. B no esq. local correspondente a rel. A na tabela de correlacionamentos:
Se sim:

Se o nome da rel. A for diferente do nome da rel. B:

Gerar transformação sem revisão do usuário:
$renRel(<< rel.\ B >>, << rel.\ A >>)$

Executar a etapa Resolução de Conflitos entre Chaves Primárias passando A e B como parâmetros.

Se não:

Executar a etapa Extração de Chaves Primárias passando A como parâmetro:

Função PróximaRelação

Objetivo: Indicar a próxima rel. a ser trabalhada na fase Unificar Chaves Primárias ou na fase Unificar Atributos.
Parâmetro de Entrada: esq. de união e fase atual.
Parâmetro de Saída: rel. do esq. de união.

Para cada rel. A do esq. de união em que haja algum att. chave não unificado (fase Unificar Chaves Primárias) ou algum att. não chave não unificado (fase Unificar Atributos):

Para cada chave estrangeira em que A for filha de uma rel. cujos atts. chaves não tenham sido unificados (fase Unificar Chaves Primárias) ou que haja atts. não chave não unificados (fase Unificar Atributos) A recebe 1 ponto.

Guardar a quantidade de pontos que A recebeu.

Devolver como resultado da função uma das rels. que tenha recebido o menor número de pontos.

B.2.3.1 Resolução de Conflitos entre Chaves Primárias

Lembrete: esta estapa recebe uma rel. A do esq. de união e uma rel. B do esq. local da etapa anterior.

Perguntar ao usuário se o conjunto de atts. chave de A e B são equivalentes.
Se sim:

Para cada atr. chave A' da rel. A ainda não unificado:

Perguntar ao usuário qual é o atr. chave B' da rel. B equivalente ao att. A' da rel. A.

Se o nome do att. A' for diferente do nome do att. B':

Gerar transformação sem revisão do usuário:
$renAtt(<<$ *att. B'* $>>, <<$ *att. A'* $>>)$

Unificar A' e B'.

Se não:

Para cada atr. chave A' da rel. A ainda não unificado:

Pedir ao usuário para indicar, caso exista, um atr. B' na rel. B que seja equivalente ao atr. A' da rel. A.
Se sim:.

Se o nome do att. A' for diferente do nome do att. B':

Gerar transformação sem revisão do usuário:
$renAtt(<<$ *att. B'* $>>, <<$ *att. A'* $>>)$

Unificar A' e B'.

Se não:

Pedir ao usuário para indicar, caso exista, um atr. C' de uma rel. C pertencente ao esq. local do qual possam ser extraídas as instâncias do atr. A' da rel. A.
Se sim:

Gerar transformação com revisão do usuário:
$addAtt(<<$ *rel. A, att. A'* $>>, [(x_1, \ldots, x_n, y)$
$\mid (x_1, \ldots, x_n, y) \leftarrow<<$ *rel. C, att. C'* $>>])$
(Obs: O lista de atts. $(x_1, \ldots, x_n)$ se refere aos atts. chaves de B.)

Unificar A' com o atr. criado.

Se não:

Gerar transformação com revisão do usuário:
$extAtt(<<$ *rel. B, att. A'* $>>,$ *void*$)$

Remover a restrição das chaves primárias da rel. B:
$delPK(<<$ *rel. B,* $<<$ *rel. B, lista de atrs. chaves de B* $>>>>)$

Recriar a restrição das chaves primárias da rel. B conforme a rel. A:
$delPK(<<$ *rel. A,* $<<$ *rel. A, lista de atrs. chaves de A* $>>>>)$

B.2.3.2 Extração de Chaves Primárias

Lembrete: esta estapa recebe uma rel. A do esq. de união da etapa anterior.

Perguntar ao usuário se é possível obter as instâncias do conjunto de atts. chaves através do esq. local ou através de valores fixos.
Se sim:

Pedir para o usuário indicar, se existir, a rel. B no esq. local do qual se vai extrair o conunto de atts. chaves:

Gerar transformação com revisão do usuário:
$addRel(<<\ rel.\ A\ >>, [(x_1, \ldots, x_n) \mid (x_1, \ldots, x_n) \leftarrow<<\ rel.\ B\ >>])$
(Obs: O valor n é o número de atts. chaves da rel. A.)

Para cada att. chave A' da rel. A:

Gerar transformação sem revisão do usuário:
$addAtt(<<\ rel.\ A,\ att.\ A'\ >>, [(x_1, \ldots, x_k, \ldots, x_n, x_k)$
$\mid (x_1, \ldots, x_n) \leftarrow<<\ rel.\ A\ >>])$
(Obs: O valor k é um possível valor de n relativo ao att. A'.)

Unificar A' com o atr. criado.

Gerar transformação sem revisão do usuário:
$addPK(<<\ rel.\ A <<\ rel.\ A,\ atts.\ chave\ da\ rel.\ A\ >>>>)$

Se não:

Gerar transformação sem revisão do usuário:
$extRel(<<\ rel.\ A\ >>,\ void\ )$

Para cada att. A' chave ou não chave da rel. A:

Gerar transformação sem revisão do usuário:
$extAtt(<<\ rel.\ A\ ,\ att.\ A'\ >>,\ void\ )$

Gerar transformação sem revisão do usuário:
$addPK(<<\ rel.\ A\ ,<<\ atts.\ chaves\ de\ A\ >>>>)$

Alterar a entrada na tabela de correlacionamentos de modo que a entrada relativa a rel. A aponte para a rel. criada.

B.2.4 Unificar Atributos

Enquanto existirem rels. no esq. de união com atts. não unificados:

Aplicar a função PróximaRelação para escolher uma rel. A do esq. de união e tomar como rel. B a rel. do esq. local correspondente na tabela de correlacionamentos.

Para cada att. não unificado A' da rel. A do esq. de união:

Pedir para o usuário indicar, se existir, um att. B' na rel. B do esq. local que seja equivalente a A'.
Se sim:

Se o nome do att. A' for diferente do nome do att. B':

Gerar transformação sem revisão do usuário:
$renAtt(<<\ att.\ B'\ >>, <<\ att.\ A'\ >>)$

Unificar A' e B'.

Se não:

Pedir para o usuário indicar, se existir, um att. C' de uma rel. C do esq. local de onde possam ser extraídas as instâncias do atr. A' da rel. A.
Se sim:

Gerar transformação com revisão do usuário:
$addAtt(<<\ rel\ A,\ att.\ A'\ >>, [(x_1, \ldots, x_n, y)$
$|\ (x_1, \ldots, x_n, y) \leftarrow <<\ rel.\ C,\ att.\ C'\ >>])$

Unificar A' com o atr. criado.

Se não:

Gerar transformação sem revisão do usuário:
$extAtt(<<\ rel\ A,\ att.\ A'\ >>,\ void\ )$

Unificar A' com o atr. criado.

B.2.5 Remover Objetos não Unificados

Para cada rel. A do esq. local com atts. não chaves não unificados:

Para cada att. A' da rel. A que não tenha sido unificado:

Tentar encontrar nas transformações geradas uma transformação reversa para o att. A'.
Se sim:

Gerar transformação com revisão do usuário:
delAtt(<< *rel A, att. A'* >>, *transformação reversa*)

Se não:

Gerar transformação com revisão do usuário:
extAtt(<< *rel A, att. A'* >>, *void*)

Para cada rel. A do esq. local com atts. chaves não unificados:

Gerar transformação sem revisão do usuário:
delPK(<< *rel. A* << *rel. A, atts. chaves da rel. A* >>>>)

Para cada att. chave A' da rel. A que não tenha sido unificado:

Gerar transformação sem revisão do usuário:
delAtt(<< *rel. A, att. A'* >>, $[(x_1, \ldots, x_k, \ldots, x_n, x_k)$
$| \ (x_1, \ldots, x_n) \leftarrow$<< *rel. A* >>$])$
(Obs: O valor k é um possível valor de n relativo ao att. A'.)

Tentar encontrar nas transformações geradas uma transformação reversa para a rel. A.
Se sim:

Gerar transformação com revisão do usuário:
delRel(<< *rel A* >>, *transformação reversa*)

Se não:

Gerar transformação com revisão do usuário:
extRel(<< *rel A* >>, *void*)

B.2.6 Gerar Chaves Estrangeiras

Para cada chave estrangeira com uma rel. filha A e uma rel. pai B no esq. de união:

Gerar transformação sem revisão do usuário:
addFK(<< *rel. A,* <<*rel. A, atrs. da chave estrangeira na rel. A*>>,
rel. B, <<*rel. B, atrs. da chave estrangeira na rel. B*>>>>)
